Machine Learning: Models And Algorithms

Daniel Bloch

28th of December 2018

The copyright to this computer software and documentation is the property of Quant Finance Ltd. It may be used and/or copied only with the written consent of the company or in accordance with the terms and conditions stipulated in the agreement/contract under which the material has been supplied.

Copyright © 2018 Quant Finance Ltd
Quantitative Analytics, London

Electronic copy available at: https://ssrn.com/abstract=3307566

Machine Learning: Models And Algorithms

Daniel BLOCH [1]
QUANT FINANCE
Working Paper

28th of December 2018
Revised version : 1.0.1

[1] dan@quantfin.eu

Electronic copy available at: https://ssrn.com/abstract=3307566

Abstract

This textbook introduces the mathematical models and algorithms utilised in machine learning, covering supervised and unsupervised learning as well as reinforcement learning: In supervised learning we present artificial neural networks, recurrent neural networks and associative reservoir computing, while in unsupervised learning we present radial basis function (RBF), recurrent RBF networks amongst others. In both cases we give examples for chaotic series prediction as well as financial time series. We then introduce the learning theory and formalise supervised learning in the cases of neural networks and recurrent neural networks. Using stochastic control theory and dynamic programming we introduce reinforcement learning (RL) and deep RL, presenting various algorithms for learning the optimal policy. Then, we introduce constrained optimisation and discuss the use of machine learning for solving some continuous and discrete optimisation problems. Finally, we apply supervised and reinforcement learning to the pricing and hedging of option prices.

Machine Learning, Supervised Learning, Unsupervised Learning, Reinforcement Learning, Artificial Neural Networks, Recurrent Neural Networks, Associative Reservoir Computing, Constrained Optimisation, Stochastic Control, Dynamic Programming, Option Pricing

Electronic copy available at: https://ssrn.com/abstract=3307566

I would like to thank my wife and children for their patience and support during this adventure.

Electronic copy available at: https://ssrn.com/abstract=3307566

Contents

I Knowledge representation **15**

1 Presentation **16**

1.1 Models . . . 16
1.1.1 Mathematical models . . . 16
1.1.2 Algorithms . . . 17
1.1.2.1 An overview . . . 17
1.1.2.2 Description . . . 18
1.1.3 Machine learning . . . 19
1.1.3.1 An overview . . . 19
1.1.3.2 Description . . . 20
1.1.3.3 Some properties . . . 21
1.1.3.3.1 Prediction . . . 21
1.1.3.3.2 Optimisation . . . 21
1.1.3.3.3 Generalisation . . . 21
1.1.3.3.4 Limitation . . . 22
1.1.3.4 Some models . . . 22
1.1.3.5 Applications to the industry . . . 23
1.2 Neural networks . . . 24
1.2.1 An overview . . . 24
1.2.1.1 A brief history . . . 24
1.2.1.2 A Description . . . 24
1.2.2 Modelling changing environment . . . 25
1.2.2.1 Multiple characteristic timescales . . . 25
1.2.2.2 Long range dependence . . . 26
1.2.2.3 The variable weight neural network . . . 27
1.2.3 Interpreting neural networks . . . 27
1.2.3.1 Explaining the input feature importance . . . 27
1.2.3.2 Determining the contribution of explanatory variables . . . 28
1.2.4 Optimisation for machine learning . . . 28
1.2.4.1 Function estimation . . . 28
1.2.4.2 Parameters estimation . . . 29
1.3 The learning process . . . 30
1.3.0.1 Prediction tasks . . . 30
1.3.0.2 Training and exploitation of the model . . . 31
1.3.0.3 The framework . . . 32
1.3.0.3.1 The input variables . . . 32
1.3.0.3.2 The data . . . 33
1.3.0.3.3 Evaluation methods . . . 33

Electronic copy available at: https://ssrn.com/abstract=3307566

1.3.0.3.4 The measures . . . 33
1.3.1 Using neural networks to model returns . . . 33
1.3.1.1 An overview . . . 33
1.3.1.2 Combining neural networks with evolutionary algorithms . . . 34

2 Data, information and knowledge **35**
2.1 Introduction to information theory . . . 35
2.1.1 Presenting a few concepts . . . 35
2.1.1.1 Uncertainty . . . 35
2.1.1.2 Coding . . . 35
2.1.1.3 Randomness . . . 36
2.1.2 Some facts on entropy in information theory . . . 37
2.1.3 Relative entropy and mutual information . . . 39
2.1.4 Bounding performance measures . . . 40
2.1.5 Feature selection . . . 42
2.2 Introduction to data mining . . . 45
2.2.1 From data to information . . . 45
2.2.1.1 Data . . . 45
2.2.1.1.1 Definitions . . . 45
2.2.1.1.2 Data set . . . 46
2.2.1.2 Data mining . . . 46
2.2.1.3 Modelling . . . 47
2.2.2 Classification . . . 48
2.2.2.1 Terminology . . . 48
2.2.2.2 Definitions . . . 48
2.2.2.3 The algorithms . . . 49
2.2.3 The challenges of computational learning . . . 50
2.2.4 Evaluation and validation . . . 52

3 Ensemble models **55**
3.1 The base models . . . 55
3.1.1 Introduction . . . 55
3.1.1.1 Presentation . . . 55
3.1.1.2 Definitions . . . 56
3.1.1.3 Diversity . . . 57
3.1.1.4 Generalisation error . . . 57
3.1.2 Some simple models . . . 58
3.1.2.1 Decision tree . . . 58
3.1.2.2 Boosting . . . 59
3.1.2.3 Bagging . . . 61
3.1.3 Random forest . . . 61
3.1.3.1 Tree bagging . . . 61
3.1.3.2 The procedure . . . 62
3.1.4 Towards combination methods . . . 64
3.1.4.1 Averaging . . . 64
3.1.4.1.1 Simple averaging . . . 64
3.1.4.1.2 Weighted averaging . . . 65
3.1.4.2 Voting . . . 66
3.1.4.2.1 Majority voting . . . 66
3.1.4.2.2 Plurality voting . . . 67

Electronic copy available at: https://ssrn.com/abstract=3307566

3.1.4.2.3 Weighted voting . . . 67
3.1.4.2.4 Soft voting . . . 68
3.2 Online learning and regret-minimising algorithms . . . 69
3.2.1 Simple online algorithms . . . 69
3.2.1.1 The Halving algorithm . . . 69
3.2.1.2 The weighted majority algorithm . . . 69
3.2.2 The online convex optimisation . . . 70
3.2.2.1 The online linear optimisation problem . . . 70
3.2.2.2 Considering Bergmen divergence . . . 71
3.2.2.3 More on the online convex optimisation problem . . . 72
3.3 Combining by learning . . . 73
3.3.1 Stacking . . . 73
3.3.2 Infinite ensemble . . . 75
3.3.3 Other methods . . . 76
3.3.3.1 The algebraic methods . . . 76
3.3.3.2 Mixture of experts . . . 76
3.3.4 Diversity measures . . . 77
3.3.4.1 Pairwise measures . . . 78
3.3.4.2 Non-pairwise measures . . . 79

II Neural Networks 81

4 Introduction to artificial neural networks 82
4.1 The classic networks . . . 82
4.1.1 The Hopfield networks . . . 83
4.1.1.1 Description . . . 83
4.1.1.2 The energy function . . . 85
4.1.2 The multilayer perceptron . . . 86
4.1.2.1 Description . . . 86
4.1.2.2 The universal approximation theorem . . . 86
4.1.3 Gradient descent and the delta rule . . . 87
4.2 Introducing multilayer networks . . . 89
4.2.1 Describing the problem . . . 89
4.2.2 Describing the algorithm . . . 90
4.2.3 Describing the nonlinear transformation . . . 90
4.2.4 A simple example . . . 92
4.3 Multi-layer back propagation . . . 93
4.3.1 The output layer . . . 93
4.3.2 The first hidden layer . . . 94
4.3.3 The next hidden layer . . . 96
4.3.4 Some remarks . . . 98
4.4 Summarising the feedforward ANN . . . 99
4.4.1 Forward pass . . . 99
4.4.2 Backward pass . . . 100

Electronic copy available at: https://ssrn.com/abstract=3307566

5 Recurrent neural networks **103**
5.1 Presenting recurrent neural networks . . . 103
5.1.1 An overview . . . 103
5.1.2 The algorithm . . . 104
5.1.2.1 Forward pass . . . 104
5.1.2.2 Backward pass . . . 105
5.2 The long short-term memory . . . 107
5.2.1 The vanishing gradient problem . . . 107
5.2.1.1 Description . . . 107
5.2.1.2 The constant error carousel . . . 108
5.2.2 Network architecture . . . 108
5.2.3 The learning algorithm . . . 113
5.2.3.1 Computational Complexity of LSTM . . . 113
5.3 Reservoir computing . . . 114
5.3.1 Describing the Reservoir methods . . . 114
5.3.1.1 Description . . . 114
5.3.1.2 The parameters . . . 115
5.3.1.3 The readout . . . 116
5.3.1.4 Some comments . . . 117
5.3.2 Choosing the parameters . . . 118
5.3.2.1 Input Scaling . . . 118
5.3.2.2 Size of the reservoir . . . 118
5.3.2.3 Sparsity . . . 119
5.3.2.4 Spectral Radius . . . 119
5.3.2.5 Leak Rate . . . 119
5.3.3 Some improvements . . . 119
5.3.4 Empirical results . . . 120
5.4 Other Models . . . 128
5.4.1 Variable weights neural networks . . . 128
5.4.1.1 Description . . . 128
5.4.1.2 Application to finance . . . 128

6 Towards dynamically stable reservoirs **130**
6.1 A specific implementation . . . 130
6.1.1 Overview of the topology . . . 130
6.1.2 Cost function . . . 131
6.1.3 Random generation of the input and feedback matrices . . . 131
6.1.4 Topology of the reservoir matrix . . . 131
6.1.5 Reservoir dynamics . . . 132
6.2 Optimisation methods . . . 133
6.2.1 Random Generation: GEN . . . 135
6.2.2 Stochastic Gradient Descent: SGD . . . 136
6.3 Adaptive filters for online learning . . . 138
6.3.1 A review of adaptive filtering theory . . . 138
6.3.1.1 Simple gradient descent . . . 140
6.3.1.2 Recursive least squares . . . 140
6.3.2 Online algorithm . . . 141
6.4 Some results . . . 141
6.4.1 Using Offline algorithms . . . 141
6.4.1.1 The Mackey-Glass series . . . 141

Electronic copy available at: https://ssrn.com/abstract=3307566

6.4.1.2 Financial time series . . . 143
6.4.2 Using online algorithms . . . 144
6.4.2.1 The Mackey-Glass series . . . 144
6.4.2.2 Financial time series . . . 147
6.5 Associative Reservoir Computing . . . 149
6.5.1 Paradigm . . . 149
6.5.1.1 Definitions . . . 149
6.5.1.2 The model . . . 149
6.5.1.3 Attractor-based computation . . . 150
6.5.2 States stabilisation techniques . . . 151
6.5.2.1 Reservoir dynamics calibration to observable stable states . . . 152
6.5.2.2 Learning with filtered inputs . . . 152

7 Introduction to unsupervised learning 154
7.1 Batch intrinsic plasticity . . . 154
7.1.1 Description . . . 154
7.1.2 Some results . . . 157
7.2 Radial basis function . . . 158
7.2.1 Description . . . 158
7.2.2 Some results . . . 161
7.3 Recurrent RBF network . . . 166
7.3.1 Standard RRBFN . . . 166
7.3.1.1 Pseudo code . . . 166
7.3.1.2 Parameter tuning . . . 167
7.3.2 Randomly parametrised RRBFN . . . 168
7.3.3 Directional parametrised RRBFN . . . 169
7.3.4 Empirical results . . . 169
7.3.5 Unsupervised learning for RRBFN . . . 170
7.3.5.1 Pseudo code . . . 172
7.3.5.2 Empirical results . . . 173

III The mathematical formalism 179

8 Supervised learning 180
8.1 Introduction to learning theory . . . 180
8.1.1 Function estimation . . . 180
8.1.1.1 Risk minimisation . . . 180
8.1.1.2 Empirical risk minimisation . . . 181
8.1.1.2.1 Convergence . . . 181
8.1.1.2.2 Structural risk minimisation . . . 182
8.1.1.2.3 Bias-variance tradeoff . . . 182
8.1.2 Solving the optimisation problem . . . 182
8.1.2.1 The hypothesis function . . . 182
8.1.2.2 The loss function . . . 184
8.1.2.3 Regularisation and stability . . . 185
8.1.2.4 Solving for the optimum . . . 186
8.1.2.4.1 The batch gradient descent . . . 186
8.1.2.4.2 The stochastic gradient descent . . . 186
8.1.2.5 Online convex optimisation . . . 187

Electronic copy available at: https://ssrn.com/abstract=3307566

8.1.3 The dynamical system . . . 187
8.1.4 Handling memory . . . 189
8.2 SVM models . . . 190
8.2.1 An optimisation problem . . . 190
8.2.2 The algorithms . . . 192
8.2.2.1 Online algorithms . . . 192
8.2.2.1.1 A simple algorithm . . . 192
8.2.2.1.2 More complex algorithms . . . 193
8.2.2.1.3 Parallel algorithm . . . 193
8.2.2.2 Offline algorithms . . . 194
8.2.2.2.1 The Lagrange multiplier . . . 194
8.2.2.2.2 The kernel trick . . . 195
8.2.3 Reproducing kernel Hilbert spaces . . . 196
8.2.3.1 The Hilbert spaces . . . 197
8.2.3.2 Nonparametric regression . . . 198
8.3 Forecasting models . . . 198
8.3.1 Regression models . . . 198
8.3.1.1 Data normalisation . . . 198
8.3.1.2 Description . . . 199
8.3.1.3 Linear regression . . . 200
8.3.2 The need to forecast volatility . . . 201
8.3.2.1 Presentation . . . 201
8.3.2.2 A first decomposition . . . 201
8.3.2.3 The structure of volatility models . . . 202
8.3.2.3.1 Presentation . . . 202
8.3.2.3.2 Benchmark volatility models . . . 203
8.3.3 Generalised nonlinear nonparametric models . . . 204
8.3.3.1 Presentation . . . 204
8.3.3.2 Describing the models . . . 204

9 Reinforcement learning 207
9.1 Presentation . . . 207
9.2 Learning with stationary reward . . . 208
9.2.1 Description . . . 209
9.2.2 The REINFORCE algorithm . . . 209
9.2.2.1 Some examples . . . 211
9.2.2.1.1 Bernoulli neurons . . . 211
9.2.2.1.2 Binary LN neurons . . . 211
9.3 Markov decision process . . . 212
9.3.1 The problem . . . 213
9.3.1.1 The transition probabilities . . . 213
9.3.1.2 The average reward . . . 215
9.3.2 The Bellman equations . . . 217
9.3.3 A mathematical formalism . . . 218
9.3.3.1 The value function . . . 219
9.3.3.1.1 The optimal policy . . . 220
9.3.3.2 The Q-value function . . . 221
9.4 Learning the optimal policy . . . 223
9.4.1 The policy iteration . . . 224
9.4.1.1 Introduction . . . 224

Electronic copy available at: https://ssrn.com/abstract=3307566

9.4.1.2 Value Iteration . . . 225
9.4.1.3 TD learning . . . 225
9.4.1.4 Fitted value iteration . . . 226
9.4.2 The Q-learning . . . 227
9.4.2.1 Q-learning . . . 227
9.4.2.2 Deep Q-learning . . . 228
9.4.2.3 Experience replay . . . 229
9.4.2.4 The continuous state-action method . . . 230
9.4.3 The policy gradients . . . 230
9.4.3.1 The REINFORCE algorithm . . . 231
9.4.3.1.1 Example . . . 233
9.4.3.2 Variance reduction . . . 233
9.4.3.2.1 Some solutions . . . 233
9.4.3.2.2 Baseline function . . . 234
9.4.4 The Actor-Critic algorithm . . . 234
9.4.4.1 Description . . . 234
9.4.4.1.1 Policy evaluation . . . 235
9.4.4.2 The algorithm . . . 236
9.4.4.3 State-dependent baselines . . . 237
9.4.5 Policy gradient improvement . . . 240
9.4.5.1 Policy iteration . . . 240
9.4.5.2 An optimisation problem . . . 242
9.4.5.2.1 Recovering the natural policy gradient . . . 243

IV Optimisation **245**

10 Introduction to constrained optimisation **246**
10.1 Introduction . . . 246
10.1.1 Defining the problem . . . 246
10.1.1.1 The constrained problem . . . 246
10.1.1.2 Some examples . . . 247
10.1.2 Conditions for a local optimum . . . 248
10.1.2.1 Local minimum . . . 248
10.1.2.2 Local maximum . . . 248
10.1.3 Conditions for constrained local optimum . . . 248
10.1.3.1 The equality constraint . . . 249
10.1.3.2 The inequality constraint . . . 249
10.1.4 Conditions for a global optimum . . . 250
10.1.5 Some solutions . . . 250
10.2 Unconstrained optimisation . . . 251
10.2.1 Some classical methods . . . 251
10.2.1.1 The gradient descent . . . 251
10.2.1.2 The subgradient descent . . . 252
10.2.1.3 The Newton's method . . . 253
10.2.1.4 The convergence . . . 253
10.2.2 Adding constraints . . . 253
10.2.2.1 The projected subgradient descent . . . 253
10.2.2.2 The subgradient descent . . . 254
10.2.2.3 The Newton method . . . 254

Electronic copy available at: https://ssrn.com/abstract=3307566

10.3 A first approach to constrained optimisation . . . 254
10.3.1 Penalty and barriers . . . 254
10.3.1.1 Log barrier method . . . 255
10.3.1.2 Central path . . . 255
10.3.1.3 Squared penalty method . . . 256
10.3.2 The augmented Lagrangian approach . . . 256
10.3.2.1 The equality constraint . . . 256
10.3.2.2 The inequality constraint . . . 257
10.4 The Lagrangian approach . . . 258
10.4.1 The KKT conditions . . . 258
10.4.2 Implication from the KKT conditions . . . 259
10.4.2.1 Equality constraints . . . 259
10.4.2.2 Inequality constraints . . . 260
10.4.3 The Lagrangian dual problem . . . 260
10.4.3.1 Definition . . . 260
10.4.3.2 Relation between primal and dual solutions . . . 261
10.4.3.3 Interpretation . . . 261
10.4.3.4 Summary . . . 262
10.4.3.5 Equivalence . . . 263
10.4.4 Additional . . . 264
10.4.4.1 The log barrier method revisited . . . 264
10.4.4.2 Non-convex optimisation . . . 264
10.5 Standard convex programs . . . 265
10.5.1 Simple programming models . . . 265
10.5.1.1 Quadratic programming . . . 265
10.5.1.2 Linear programming . . . 266
10.5.1.3 Second-order cone programming . . . 266
10.5.1.4 Semidefinite programming . . . 266
10.5.2 Some examples . . . 266
10.5.2.1 Quadratic programming . . . 266
10.5.2.2 The Lasso problem . . . 267
10.6 Generalising the penalty function approach . . . 267
10.6.1 The energy function . . . 267
10.6.2 The penalty functions . . . 268
10.6.2.1 Type I . . . 268
10.6.2.2 Type II . . . 269
10.6.2.3 Type III . . . 269

11 Global search optimisation **271**
11.1 Evolutionary algorithms . . . 271
11.1.1 A brief history of evolutionary algorithms . . . 271
11.1.2 Some optimisation methods . . . 272
11.1.2.1 Random optimisation . . . 272
11.1.2.2 Particle swarm optimisation . . . 273
11.1.2.3 Cross entropy optimisation . . . 274
11.1.2.4 Simulated annealing . . . 275
11.1.2.5 Introduction to genetic algorithms . . . 276
11.1.3 Introduction to differential evolution . . . 277
11.1.3.1 The DE algorithm . . . 277
11.1.3.1.1 The mutation . . . 277

Electronic copy available at: https://ssrn.com/abstract=3307566

11.1.3.1.2 The recombination . . . 278
11.1.3.1.3 The selection . . . 279
11.1.3.1.4 Simple convergence criteria . . . 279
11.1.3.2 Pseudocode . . . 280
11.1.3.3 The strategies . . . 280
11.1.3.3.1 Scheme DE1 . . . 280
11.1.3.3.2 Scheme DE2 . . . 281
11.1.3.3.3 Scheme DE3 . . . 281
11.1.3.3.4 Scheme DE4 . . . 281
11.1.3.3.5 Scheme DE5 . . . 282
11.1.3.3.6 Scheme DE6 . . . 282
11.1.3.3.7 Scheme DE7 . . . 283
11.1.3.3.8 Scheme DE8 . . . 283
11.1.3.4 Improvements . . . 283
11.1.3.4.1 The tuning parameters . . . 284
11.1.3.4.2 Ageing . . . 284
11.1.3.4.3 Constraints on parameters . . . 284
11.1.3.4.4 Convergence . . . 285
11.1.3.4.5 Self-adaptive parameters . . . 285
11.1.3.4.6 Selection . . . 286
11.1.3.5 Convergence criteria revised . . . 286
11.2 Handling the constraints . . . 289
11.2.1 Describing the problem . . . 289
11.2.2 Defining the feasibility rules . . . 290
11.2.3 Improving the feasibility rules . . . 290
11.2.4 Handling diversity . . . 292
11.3 The proposed algorithm . . . 292

12 Solving optimisation problems with NNs **294**
12.1 Hybrid intelligent modelling . . . 294
12.1.1 ANN plus GA in finance . . . 294
12.1.1.1 An overview . . . 294
12.1.1.2 Some examples . . . 295
12.1.1.2.1 Feature selection . . . 295
12.1.1.2.2 Weight selection . . . 295
12.1.2 ANN plus DE in finance . . . 296
12.2 Solving constrained optimisation problems . . . 297
12.2.1 The method . . . 297
12.2.1.1 A short review . . . 297
12.2.1.2 Mapping optimisation problems to networks . . . 298
12.2.2 Some optimisation models . . . 298
12.2.2.1 Discrete synchronous Hopfield network . . . 298
12.2.2.2 Discrete asynchronous Hopfield network . . . 298
12.2.2.3 Analog Hopfield network . . . 299
12.2.2.4 Boltzmann machine . . . 299
12.2.2.5 Cauchy machine . . . 299
12.2.2.6 Hybrid Scheme . . . 300
12.2.3 Approximation with non-linear regression . . . 301
12.2.3.1 The method . . . 301
12.2.3.2 Defining the network . . . 301

Electronic copy available at: https://ssrn.com/abstract=3307566

12.2.3.3 The Lagrangian method . . . 302
12.3 Solving combinatorial optimisation problems . . . 304
12.3.1 Defining the problem . . . 304
12.3.1.1 Description . . . 304
12.3.1.2 Formalising the problem . . . 304
12.3.2 Solving the TSP . . . 305
12.3.2.1 A review . . . 305
12.3.2.2 The problem . . . 305
12.3.3 The REINFORCE algorithm . . . 306
12.3.4 Applying reinforcement learning . . . 307

V Applications **310**

13 Mathematical Finance **311**
13.1 Option pricing . . . 311
13.1.1 The framework . . . 311
13.1.1.1 Defining the replicating portfolio . . . 312
13.1.1.2 The BS-formula . . . 314
13.1.1.3 The implied volatility . . . 315
13.1.1.4 The no-arbitrage conditions . . . 317
13.1.1.5 Characterising the smile dynamics . . . 317
13.1.1.5.1 The regimes of volatility . . . 317
13.1.1.5.2 Space homogeneity versus space inhomogeneity . . . 318
13.1.2 The calibration problem . . . 319
13.1.2.1 The general idea . . . 319
13.1.2.1.1 Description . . . 319
13.1.2.1.2 Measures of pricing errors . . . 319
13.1.2.1.3 Moments estimation . . . 320
13.1.2.2 The problem . . . 320
13.1.2.2.1 The choice of a volatility model . . . 320
13.1.2.2.2 Accounting for no-arbitrage . . . 320
13.1.2.2.3 The standard approach . . . 321
13.1.2.2.4 A global optimum . . . 321
13.1.3 Supervised learning . . . 321
13.1.3.1 Fitting model prices . . . 322
13.1.3.1.1 The tests . . . 322
13.1.3.2 Fitting model prices . . . 323
13.1.3.2.1 The tests . . . 323
13.1.3.2.2 Results . . . 323
13.1.3.2.3 Analysis and solutions . . . 324
13.1.3.3 Completing market prices . . . 324
13.1.3.3.1 The problem . . . 324
13.1.3.3.2 Generating arbitrage-free data . . . 324
13.1.4 Reinforcement learning . . . 325
13.1.4.1 Introduction . . . 325
13.1.4.1.1 Problems with continuous models . . . 325
13.1.4.1.2 Accounting for risk . . . 326
13.1.4.2 Deterministic BS-model . . . 327
13.1.4.2.1 Pricing in discrete time . . . 327

Electronic copy available at: https://ssrn.com/abstract=3307566

13.1.4.2.2 The value function . . . 327
13.1.4.2.3 The Q-value function . . . 329
13.1.4.3 Learning the optimal policy . . . 329
13.1.4.3.1 Known model . . . 330
13.1.4.3.2 Unknown model . . . 330
13.1.4.4 Non-normal rates model . . . 331
13.1.4.4.1 The assumption of normal returns . . . 331
13.1.4.4.2 Beyond normal returns . . . 331
13.1.4.4.3 The modified risk-averse price . . . 331
13.1.4.4.4 The modified value function . . . 332
13.2 Appendices . . . 334
13.3 Miscellaneous . . . 334
13.3.1 The replication portfolio . . . 334
13.3.1.1 Complete market . . . 334
13.3.1.2 Incomplete market . . . 334
13.3.2 Local risk minimisation . . . 335
13.3.2.1 Portfolio evaluation . . . 335
13.3.2.2 Optimal hedging . . . 336
13.3.2.3 The risk-averse price . . . 336
13.3.3 The capital asset pricing model . . . 337
13.3.3.1 Mean-variance criterion . . . 337
13.3.3.1.1 Normal returns . . . 337
13.3.3.1.2 Non-normal returns . . . 338
13.3.3.2 Markowitz solution to the portfolio allocation problem . . . 339
13.3.3.3 The expected growth rate . . . 340
13.3.4 Point on Kolmogorov-Compatibility . . . 340
13.4 The parametric MixVol model . . . 341
13.4.1 Description of the model . . . 341
13.4.2 Computing the Greeks . . . 343
13.4.3 Scenarios analysis . . . 343

VI Annexes **346**

14 Some mathematical facts **347**
14.1 Notation . . . 347
14.2 Some functions . . . 347
14.2.1 Miscellaneous . . . 347
14.2.1.1 The derivative . . . 347
14.2.1.2 The subgradient . . . 348
14.2.1.3 The kernel functions . . . 349
14.2.1.4 The Dirac function . . . 350
14.3 Some facts on convex and concave analysis . . . 351
14.3.1 Convex functions . . . 352
14.3.2 Concave functions . . . 353
14.3.3 Some approximations . . . 354
14.4 Introduction to wavelet theory . . . 354
14.4.1 Some definitions . . . 354
14.4.1.1 Continuous Wavelet . . . 355
14.4.1.2 Discrete Wavelet . . . 355

Electronic copy available at: https://ssrn.com/abstract=3307566

14.4.2 Decomposition schemes . . . 356
14.4.2.1 Discrete Wavelet Transform . . . 356
14.4.2.2 Stationary Wavelet Transform . . . 357
14.4.2.3 Wavelet Packet Transform . . . 358
14.4.3 Denoising techniques . . . 359
14.5 Some random sampling . . . 360
14.5.1 The sample moments . . . 361
14.5.2 Estimation of a ratio . . . 363
14.5.3 Stratified random sampling . . . 364
14.5.4 Geometric mean . . . 368
14.5.5 Stochastic approximation . . . 368
14.5.6 Accounting for time and earning profits . . . 369
14.5.6.1 Time factor . . . 369
14.5.6.2 Direction measures . . . 370
14.5.6.3 Time dependent direction profit . . . 370
14.6 Some notion of probabilities . . . 371
14.6.1 Events as sets and their probabilities . . . 371
14.6.1.1 Sets of events . . . 371
14.6.1.2 Probability measures . . . 372
14.6.1.3 Conditional events . . . 373
14.6.2 Defining random variables . . . 373
14.6.2.1 Discrete random variables . . . 373
14.6.2.2 Continuous random variables . . . 376
14.6.3 The characteristic function, moments and cumulants . . . 377
14.6.3.1 Definitions . . . 377
14.6.3.2 The first two moments . . . 378
14.6.4 Conditional moments . . . 379
14.6.4.1 Conditional expectation . . . 379
14.6.4.2 Conditional variance . . . 382
14.6.5 Multiple random variables . . . 384
14.6.5.1 Discrete variables . . . 384
14.6.5.2 Continuous variables . . . 385
14.6.6 Simple random walk . . . 386
14.6.7 Some limits . . . 386
14.6.8 Some Markov properties . . . 388
14.6.9 Ergodic theory . . . 390
14.7 Introduction to stochastic control . . . 391
14.7.1 Description of the problem . . . 391
14.7.2 HJB: the Markov control . . . 393
14.7.2.1 Description . . . 393
14.7.2.2 Some examples . . . 394
14.7.3 Terminal conditions . . . 395
14.8 Portfolio optimisation . . . 396
14.8.1 Dynamic programming . . . 396
14.8.1.1 Defining the problem . . . 396
14.8.1.2 Solving the HJB equation . . . 397
14.8.2 The martingale approach . . . 398
14.8.2.1 Admissible strategies . . . 399
14.8.2.2 Existence of an optimal pair . . . 400
14.8.3 Dynamic programming: Some proofs . . . 402

Electronic copy available at: https://ssrn.com/abstract=3307566

14.8.3.1 The principle of dynamic programming . . . 402
14.8.3.2 The HJB equation . . . 402
14.8.3.3 Sufficient conditions for an optimal solution . . . 403

15 Monte Carlo **404**
15.1 The dynamics of the state of the system . . . 404
15.1.1 The stochastic differential equation . . . 404
15.1.2 From Stratonovich equation to Ito equation . . . 405
15.1.3 Discretisation of the SDE . . . 406
15.1.3.1 The Euler scheem . . . 407
15.1.3.2 Justification of the Euler scheme . . . 407
15.1.4 Generating trajectories . . . 407
15.1.4.1 Multivariate normals . . . 407
15.1.4.2 One-dimension . . . 408
15.1.4.3 Multi-dimensions . . . 408
15.1.5 Construction of the Brownian bridge . . . 409
15.1.5.1 Construction . . . 409
15.1.5.2 Justification . . . 409
15.1.6 Simulating the state process . . . 410
15.2 The Monte Carlo engine . . . 411
15.2.1 The setup of the Monte Carlo engine . . . 411
15.2.2 Variance reduction techniques . . . 412
15.2.2.1 Control variates . . . 413
15.2.2.2 Perfect control variates . . . 413
15.2.2.3 Using the hedge as control variates . . . 414
15.2.2.4 Importance sampling . . . 415
15.2.2.5 Antithetic variables . . . 416
15.2.3 The speed of convergence . . . 416
15.2.3.1 The Richardson extrapolation . . . 416
15.2.3.2 The Romberg methods for Euler and Milshtein scheme . . . 417
15.2.4 Weak error for path-dependent functionals . . . 418
15.2.5 Romberg extrapolation for path-dependent functionals . . . 419
15.2.5.1 The stepwise constant Euler scheme . . . 419
15.2.5.2 The continuous Euler scheme . . . 420

Electronic copy available at: https://ssrn.com/abstract=3307566

Part I

Knowledge representation

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 1

Presentation

1.1 Models

1.1.1 Mathematical models

Models describe our beliefs about how the world functions; they explain a system and study the effects of different components to make predictions about behaviour. A mathematical model is a description of a system using mathematical concepts and language, and mathematical modelling is the process of developing a mathematical model. In general, mathematical models are composed of relationships and variables. That is, a model describes a system by a set of variables and a set of equations that establish relationships between the variables. Variables may be of many types such as real or integer numbers, boolean values or strings. While the variables represent some properties of the system, the actual model is the set of functions that describe the relations between the different variables. These relationships can be described by operators, such as algebraic operators, functions, differential operators, etc. When studying models, it is helpful to identify broad categories of models. Classification of individual models into these categories tells us immediately some of the essentials of their structure. Several classification criteria can be used for mathematical models according to their structure.

In business and engineering, mathematical models may be used to maximise a certain output. For example, the system under consideration may require certain inputs, and the relation between inputs and outputs may depend on other variables such as decision variables, state variables, exogenous variables, and random variables. Decision variables are sometimes known as independent variables. Exogenous variables are sometimes known as parameters or constants. The variables are not independent of each other as the state variables are dependent on the decision, input, random, and exogenous variables. Furthermore, the output variables are dependent on the state of the system (represented by the state variables).
Objectives and constraints of the system and its users can be represented as functions of the output variables or state variables. The objective functions will depend on the perspective of the model's user. Depending on the context, an objective function is also known as an index of performance, as it is some measure of interest to the user. Although there is no limit to the number of objective functions and constraints a model can have, using or optimizing the model becomes more involved (computationally) as the number increases.

Mathematical modelling problems are often classified into black box or white box models, according to how much a priori information on the system is available. A black-box model is a system of which there is no a priori information available, while a white-box model (also called glass box or clear box) is a system where all necessary information is available. Practically all systems are somewhere between the black-box and white-box models, so that this concept is only useful as an intuitive guide for deciding which approach to take. It is preferable to use as much a priori information as possible to make the model more accurate. If you use this information correctly, then the model will behave

Electronic copy available at: https://ssrn.com/abstract=3307566

correctly. Often the a priori information comes in forms of knowing the type of functions relating different variables. In black-box models one tries to estimate both the functional form of relations between variables and the numerical parameters in those functions. In that setting, we could try to use functions as general as possible to cover all different models. Examples of such general functions are neural networks (NNs) which usually do not make assumptions about incoming data, and the NARMAX (Nonlinear AutoRegressive Moving Average model with eXogenous inputs) algorithms which were developed as part of nonlinear system identification. The latter can be used to select the model terms, determine the model structure, and estimate the unknown parameters in the presence of correlated and nonlinear noise. The advantage of NARMAX models compared to neural networks is that it produces models that can be written down and related to the underlying process, whereas neural networks produce an approximation that is opaque (interpretability issues).

One way around, is to incorporate subjective information into a mathematical model based on intuition, experience, or expert opinion, or based on convenience of mathematical form. For instance, Bayesian statistics provides a theoretical framework for incorporating such subjectivity into a rigorous analysis: we specify a prior probability distribution (which can be subjective), and then update this distribution based on empirical data. However, while adding complexity usually improves the realism of a model, it can make the model difficult to understand and analyse, and can also pose computational problems, including numerical instability. In general, model complexity involves a trade-off between simplicity and accuracy of the model. It leads to Occam's razor principle which states that among models with roughly equal predictive power, the simplest one is the most desirable.

Any model which is not pure white-box contains some parameters that can be used to fit the model to the system it is intended to describe. If the modelling is done by an artificial neural network or other machine learning, the optimisation of parameters is called training, while the optimisation of model hyperparameters is called tuning and often uses cross-validation. In more conventional modelling through explicitly given mathematical functions, parameters are often determined by curve fitting.

1.1.2 Algorithms

1.1.2.1 An overview

The mathematics of George Boole (1847, 1854), Gottlob Frege (1879), and Giuseppe Peano (1888-1889) reduced arithmetic to a sequence of symbols manipulated by rules. Frege proposed a "formula language", that is a language written with special symbols constructed from specific symbols that are manipulated according to definite rules. The work of Frege was further simplified and amplified by Alfred North Whitehead and Bertrand Russell in their Principia Mathematica (PM) (1910-1913) (see Irvine [2003]). Turing and Post [1936] (independently) described a process of men-as-computers working on computations model. They started with an analysis of a human computer that they whittles down to a simple set of basic motions and states of mind. But Turing continued a step further and created a machine as a model of computation of numbers. A few years later, Turing expanded his analysis and described a function to be "effectively calculable" if its values can be found by some purely mechanical process (calculable by a machine).
In mathematics and computer science, an algorithm is an unambiguous specification of how to solve a class of problems. Kleene [1943] proposed the following definition (Thesis I known as the Church-Turing thesis):

Definition 1.1.1 *An algorithm describes a procedure, performable for each set of values of the independent variables, which procedure necessarily terminates and in such manner that from the outcome we can read a definite answer, "yes" or "no", to the question "is the predicate value true?".*

According to Rogers [1967], algorithms can perform calculation, data processing and automated reasoning tasks. An informal definition proposed by Stone [1972] is as follows:

Definition 1.1.2 *An algorithm is a set of rules that precisely defines a sequence of operations.*

Electronic copy available at: https://ssrn.com/abstract=3307566

As an effective method, an algorithm can be expressed within a finite amount of space and time and in a well-defined formal language for calculating a function. Starting from an initial state and initial input (perhaps empty), the instructions describe a computation that, when executed, proceeds through a finite number of well-defined successive states, eventually producing output and terminating at a final ending state. An algorithm has one or more outputs, that is, quantities which have a specified relation to the inputs (see Knuth [1973]). The transition from one state to the next is not necessarily deterministic; some algorithms, known as randomised algorithms, incorporate random input.
The concept of algorithm has existed for centuries. Babylonian clay tablets describe and employ algorithmic procedures to compute the time and place of significant astronomical events. Greek mathematicians used algorithms in, for example, the sieve of Eratosthenes for finding prime numbers and the Euclidean algorithm for finding the greatest common divisor of two numbers. A partial formalisation of what would become the modern concept of algorithm began with attempts to solve the Entscheidungsproblem (decision problem) posed by David Hilbert in 1928. Later formalisations were framed as attempts to define effective calculability or effective method. Those formalisations included the Godel-Herbrand-Kleene recursive functions of 1930, 1934 and 1935, Alonzo Church's lambda calculus of 1936, Emil Post's Formulation 1 of 1936, and Alan Turing's Turing machines (see Turing [1936-37] [1939]).

Most algorithms are intended to be implemented as computer programs. However, algorithms are also implemented by other means, such as in a biological neural network (for example, the human brain implementing arithmetic or an insect looking for food), in an electrical circuit, or in a mechanical device.
In computer systems, an algorithm is an instance of logic written in software by software developers, to be effective for the intended target computer(s) to produce output from given (perhaps null) input. An optimal algorithm, even running in old hardware, would produce faster results than a non-optimal (higher time complexity) algorithm for the same purpose, running in more efficient hardware; that is why algorithms, like computer hardware, are considered technology.

1.1.2.2 Description

For a given function multiple algorithms may exist. There are various ways of classifying algorithms, each with its own merits such as by implementation, by design paradigm (brute force, divide and conquer, search and enumeration), by field of study, by complexity and for optimisation problems. In the case of optimisation problems there is a more specific classification of algorithms; an algorithm for such problems may fall into one or more of the general categories described above as well as into one of the following:

- Linear programming: When searching for optimal solutions to a linear function bound to linear equality and inequality constraints, the constraints of the problem can be used directly in producing the optimal solutions.
- Dynamic programming: When a problem shows optimal substructures, meaning the optimal solution to a problem can be constructed from optimal solutions to subproblems, and overlapping subproblems, meaning the same subproblems are used to solve many different problem instances, a quicker approach called dynamic programming avoids recomputing solutions that have already been computed. Dynamic programming and memoization (it is an optimisation technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again.) go together. The main difference between dynamic programming and divide and conquer is that subproblems are more or less independent in divide and conquer, whereas subproblems overlap in dynamic programming.
- The greedy method: A greedy algorithm is similar to a dynamic programming algorithm in that it works by examining substructures, in this case not of the problem but of a given solution. Such algorithms start with some solution, which may be given or have been constructed in some way, and improve it by making small modifications. For some problems they can find the optimal solution while for others they stop at local optima, that is, at solutions that cannot be improved by the algorithm but are not optimum. The most popular use of greedy algorithms is for finding the minimal spanning tree where finding the optimal solution is possible with this method.

Electronic copy available at: https://ssrn.com/abstract=3307566

- The heuristic method: In optimisation problems, heuristic algorithms can be used to find a solution close to the optimal solution in cases where finding the optimal solution is impractical. These algorithms work by getting closer and closer to the optimal solution as they progress. In principle, if run for an infinite amount of time, they will find the optimal solution. Their merit is that they can find a solution very close to the optimal solution in a relatively short time. Such algorithms include local search, tabu search, simulated annealing, and genetic algorithms.

1.1.3 Machine learning

1.1.3.1 An overview

Turing was highly influential in the development of theoretical computer science, providing a formalisation of the concepts of algorithm and computation with the Turing machine, which can be considered a model of a general-purpose computer. He is widely considered to be the father of theoretical computer science and artificial intelligence (AI). During his stay at the Victoria University of Manchester, he continued his work on Computing Machinery and Intelligence (see Turing [1950]) where he addressed the problem of artificial intelligence, and proposed an experiment that became known as the Turing test, an attempt to define a standard for a machine to be called "intelligent". The idea was that a computer could be said to "think" if a human interrogator could not tell it apart, through conversation, from a human being. Turing suggested that rather than building a program to simulate the adult mind, it would be better rather to produce a simpler one to simulate a child's mind and then to subject it to a course of education. A reversed form of the Turing test is widely used on the Internet: the CAPTCHA test is intended to determine whether the user is a human or a computer. Along with other discoveries in neurobiology, information theory and cybernetics, it led researchers to consider the possibility of building an electronic brain. The first work recognised as AI was that of McCullouch et al. [1943] formal design of Turing complete artificial neurons.

Artificial intelligence (AI), is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and other animals. It is applied when a machine mimics cognitive functions that humans associate with other minds, such as learningand problem solving. A typical AI perceives its environment and takes actions that maximise its chance of successfully achieving its goals (see Russell et al. [2009]). The field of AI research was born at a workshop at Dartmouth College in 1956. Attendees Allen Newell (CMU), Herbert Simon (CMU), John McCarthy (MIT), Marvin Minsky (MIT) and Arthur Samuel (IBM) became the founders and leaders of AI research.
The field was founded on the claim that human intelligence can be so precisely described that a machine can be made to simulate it. Thus, early researchers developed algorithms that imitated step-by-step reasoning that humans use when they solve puzzles or make logical deductions.
In the early days of artificial intelligence (AI) as an academic discipline, some researchers were interested in having machines learn from data. They attempted to approach the problem with various symbolic methods, as well as what were then termed neural networks (NNs), which were mostly perceptrons (see McCullouch et al. [1943] and Rosenblatt [1958]) and other models that were later found to be reinventions of the generalised linear models of statistics (see Sarle [1994]). Thus, machine learning (ML) is a field of artificial intelligence (AI) that uses statistical techniques to give computer systems the ability to learn (progressively improve performance on a specific task) from data, without being explicitly programmed (see Arthur Samuel [1959]). However, by 1980, expert systems (a computer system that emulates the decision-making ability of a human expert) had come to dominate AI, and statistics was out of favour. Neural networks research had been abandoned by AI and computer science around the same time. Nonetheless, this line of research was continued outside the AI/CS field, as connectionism, by researchers from other disciplines including Hopfield, Rumelhart and Hinton. Their main success came in the mid-1980s with the reinvention of backpropagation (see Rumelhart et al. [1986c]). Machine learning, reorganised as a separate field, started to flourish in the 1990s. The field changed its goal from achieving artificial intelligence to tackling solvable problems of a practical nature. It shifted focus away from the symbolic approaches it had inherited from AI, and toward methods and models borrowed from statistics and probability theory. It also benefited from the increasing availability of digitised information, and the ability to distribute it via the Internet.

Electronic copy available at: https://ssrn.com/abstract=3307566

Modifying Alan Turing's proposal (see Turing [1950]), in which the question "Can machines think?" with the question "Can machines do what we (as thinking entities) can do?", Tom M. Mitchell [1997] proposed a formal definition of the algorithms studied in the machine learning field:

Definition 1.1.3 *Learning from data*
A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

Nowadays we are talking about the great decoupling (see Harari [2016]) where intelligence is decoupling from consciousness to create a new type of non-consciousness intelligence that can perform some tasks (playing chess, driving cars, diagnosing diseases or identifying people) far better than humans. While there are several ways leading to super-intelligence, only some of them pass through the straits of consciousness. For armies and corporations intelligence (performing tasks) is mandatory but consciousness is optional.

1.1.3.2 Description

Machine learning (ML) can give computers the ability to learn without being explicitly programmed (see A. Samuel [1959]). It focuses on designing, developing and analysing methods giving a computer the ability to follow a systematic process leading to producing results in difficult tasks that classical algorithms could not produce. In practice, machine learning explores the study and construction of algorithms that can learn from and make predictions on data (see Kohavi et al. [1998]). Such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs (see Bishop [2006]). Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible.

Machine learning tasks are typically classified into several broad categories:

- Supervised learning: The computer is presented with example inputs and their desired outputs, given by a teacher, and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback. Examples are classification, regression, object detection, semantic segmentation, images, captioning, among others.
- Semi-supervised learning: The computer is given only an incomplete training signal: a training set with some (often many) of the target outputs missing.
- Active learning: The computer can only obtain training labels for a limited set of instances (based on a budget), and also has to optimise its choice of objects to acquire labels for. When used interactively, these can be presented to the user for labelling.
- Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning). Examples are clustering, dimensionality reduction, feature learning, density estimation, among others.
- Reinforcement learning: Data (in form of rewards and punishments) are given only as feedback to the program's actions in a dynamic environment, such as driving a vehicle or playing a game against an opponent. The goal is to learn how to take actions in order to maximise reward.

Another categorisation of machine learning tasks arises when one considers the desired output of a machine-learned system:

Electronic copy available at: https://ssrn.com/abstract=3307566

- In classification, inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised way. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam".
- In regression, also a supervised problem, the outputs are continuous rather than discrete.
- In clustering, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.
- Density estimation finds the distribution of inputs in some space.
- Dimensionality reduction simplifies inputs by mapping them into a lower-dimensional space. Topic modelling is a related problem, where a program is given a list of human language documents and is tasked to find out which documents cover similar topics.

1.1.3.3 Some properties

1.1.3.3.1 Prediction Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction, known as predictive analytics in commercial use. These analytical models allow researchers, data scientists, engineers, and analysts to produce reliable, repeatable decisions and results and uncover hidden insights through learning from historical relationships and trends in the data.
Machine learning is closely related to computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimisation, which delivers methods, theory and application domains to the field. Machine learning is sometimes conflated with data mining, where the latter subfield focuses more on exploratory data analysis and is known as unsupervised learning (see Friedman [1998]).
Machine learning and data mining often employ the same methods and overlap significantly, but while machine learning focuses on prediction, based on known properties learned from the training data, data mining focuses on the discovery of (previously) unknown properties in the data (this is the analysis step of knowledge discovery in databases). Data mining uses many machine learning methods, but with different goals; on the other hand, machine learning also employs data mining methods as unsupervised learning or as a preprocessing step to improve learner accuracy.

1.1.3.3.2 Optimisation Machine learning also has intimate ties to optimisation: many learning problems are formulated as minimisation of some loss function on a training set of examples. Loss functions express the discrepancy between the predictions of the model being trained and the actual problem instances (for example, in classification, one wants to assign a label to instances, and models are trained to correctly predict the pre-assigned labels of a set of examples). The difference between the two fields arises from the goal of generalisation: while optimisation algorithms can minimise the loss on a training set, machine learning is concerned with minimising the loss on unseen samples (see Le Roux et al. [2012]).

1.1.3.3.3 Generalisation One of the main objective of machine learning is to generalise from its experience (see Bishop [2006], Mohri et al. [2012]). Generalisation in this context is the ability of a learning machine to perform accurately on new, unseen examples/tasks after having experienced a learning data set. The training examples come from some generally unknown probability distribution (considered representative of the space of occurrences) and the learner has to build a general model about this space that enables it to produce sufficiently accurate predictions in new cases.

The computational analysis of machine learning algorithms and their performance is a branch of theoretical computer science known as computational learning theory. Because training sets are finite and the future is uncertain, learning theory usually does not yield guarantees of the performance of algorithms. Instead, probabilistic bounds on the performance are quite common. The bias-variance decomposition is one way of quantifying generalisation error.

Electronic copy available at: https://ssrn.com/abstract=3307566

For the best performance in the context of generalisation, the complexity of the hypothesis should match the complexity of the function underlying the data. If the hypothesis is less complex than the function, then the model has underfit the data. However, if the complexity of the model is increased in response, then the training error should decrease. But if the hypothesis is too complex, then the model is subject to overfitting and generalisation will be poorer (see Alpaydin [2010]).

In addition to performance bounds, computational learning theorists study the time complexity and feasibility of learning. In computational learning theory, a computation is considered feasible if it can be done in polynomial time. There are two kinds of time complexity results. Positive results show that a certain class of functions can be learned in polynomial time. Negative results show that certain classes cannot be learned in polynomial time.

1.1.3.3.4 Limitation Effective machine learning is difficult because finding patterns is hard and often not enough training data are available. As a result, many machine-learning programs often fail to deliver the expected value. Reasons for this are numerous: lack of (suitable) data, lack of access to the data, data bias, privacy problems, badly chosen tasks and algorithms, wrong tools and people, lack of resources, and evaluation problems.
Further, machine learning approaches can suffer from different data biases. For instance, a machine learning system trained on one current customers only may not be able to predict the needs of new customer groups that are not represented in the training data. In addition, when trained on man-made data, machine learning is likely to pick up the same constitutional and unconscious biases already present in society.

1.1.3.4 Some models

There is a long list of machine learning algorithms such as

- decision tree learning: It uses a decision tree as a predictive model, which maps observations about an item to conclusions about the item's target value.
- association rule learning: is a method for discovering interesting relations between variables in large databases.
- artificial neural networks: It is a learning algorithm that is vaguely inspired by biological neural networks. Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modelling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables.
- deep learning: It consists of multiple hidden layers in an artificial neural network. This approach tries to model the way the human brain processes light and sound into vision and hearing. Some successful applications of deep learning are computer vision and speech recognition.
- inductive learning programming: It is an approach to rule learning using logic programming as a uniform representation for input examples, background knowledge, and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesized logic program that entails all positive and no negative examples. Inductive programming is a related field that considers any kind of programming languages for representing hypotheses (and not only logic programming), such as functional programs.
- support vector machines: They are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.
- Bayesian networks: It is a probabilistic graphical model that represents a set of random variables and their conditional independencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent

Electronic copy available at: https://ssrn.com/abstract=3307566

the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms exist that perform inference and learning.

- representation learning: Several learning algorithms, mostly unsupervised learning algorithms, aim at discovering better representations of the inputs provided during training. Classical examples include principal components analysis and cluster analysis. Representation learning algorithms often attempt to preserve the information in their input but transform it in a way that makes it useful, often as a pre-processing step before performing classification or predictions, allowing reconstruction of the inputs coming from the unknown data generating distribution, while not being necessarily faithful for configurations that are implausible under that distribution.

- rule-based machine learning: It is a general term for any machine learning method that identifies, learns, or evolves rules to store, manipulate or apply, knowledge. The defining characteristic of a rule-based machine learner is the identification and utilization of a set of relational rules that collectively represent the knowledge captured by the system. This is in contrast to other machine learners that commonly identify a singular model that can be universally applied to any instance in order to make a prediction. Rule-based machine learning approaches include learning classifier systems, association rule learning, and artificial immune systems.

Note, the neural network (NN) itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

1.1.3.5 Applications to the industry

As discussed in Section (1.1.3), machine learning is about programming computers to learn and to improve automatically with experience (see A. Samuel [1959]). While computers can not yet learn as well as people, algorithms (see Section (1.1.2)) have been devised that are effective for certain types of learning tasks, and a theoretical understanding of learning has emerged. Computer programs developed, exhibiting useful types of learning, especially in speech recognition and data mining (see Mitchell [1997]). Further, new techniques combining elements of learning, evolution and adaptation from the field of Computational Intelligence developed, capable of improving the work process and adding value to the industry by solving key business problems.

As a result, artificial intelligence (AI) is becoming an emerging asset to the workforce, and machine learning (ML) is growing in its reputation among business leaders. The latter is found in multiple industries, transforming the way business and societies operate. It will allow society to redefine efficiency. Some of the vertical breakdowns are

- Enterprise intelligence
- Enterprise functions
- Autonomous systems
- Healthcare
- Technology stack
- Energy
- Industry (agriculture, marketing, law, logistics, retail, finance, transport)

Even though segmenting market is qualitatively messy, it might clear up in time. No one knows what will take off, and this influences the behaviour of the industry. Nonetheless, Healthcare, Marketing and Finance consistently appear as areas of ML focus.

Electronic copy available at: https://ssrn.com/abstract=3307566

To illustrate some of the models and algorithms that we will present in this book we will draw examples from the Finance industry. This is because subjects such as Neural Networks, Swarm Intelligence, Fuzzy Systems and Evolutionary Computation can be applied to financial markets in a variety of ways such as predicting the future movement of stock's price or optimising a collection of investment assets (funds and portfolios). These techniques assume that there exist patterns in stock returns and that they can be exploited by analysis of the history of stock prices, returns, and other key indicators (see Schwager [1996]). With the fast increase of technology in computer science, new techniques can be applied to financial markets in view of developing applications capable of automatically manage a portfolio. Consequently, there is substantial interest and possible incentive in developing automated programs that would trade in the market much like a technical trader would, and have it relatively autonomous. Mechanical trading systems (MTS) founded on technical analysis developed. It is a mathematically defined algorithm designed to help the user make objective trading decisions based on historically reoccurring events.

1.2 Neural networks

1.2.1 An overview

1.2.1.1 A brief history

In the late 1940s, Hebb [1949] created a learning hypothesis based on the mechanism of neural plasticity that became known as Hebbian learning. It is unsupervised learning and evolved into models for long term potentiation. Researchers started applying these ideas to computational models in 1948 with Turing's B-type machines.
McCulloch and Pitts [1943] created a computational model for neural networks based on mathematics and algorithms called threshold logic. This model paved the way for neural network research to split into two approaches. One approach focused on biological processes in the brain, while the other focused on the application of neural networks to artificial intelligence. This work led to work on nerve networks and their link to finite automata (see Kleene [1956]).
Farley et al. [1954] first used computational machines, then called "calculators", to simulate a Hebbian network. Other neural network computational machines were created by Rochester et al. [1956].
Rosenblatt [1958] created the perceptron, an algorithm for pattern recognition. With mathematical notation, Rosenblatt described circuitry not in the basic perceptron, such as the exclusive-or circuit that could not be processed by neural networks at the time (see Werbos [1974]).
In 1959, a biological model proposed by Nobel laureates Hubel and Wiesel was based on their discovery of two types of cells in the primary visual cortex: simple cells and complex cells (see Hubel et al. [2005]). The first functional networks with many layers were published by Ivakhnenko et al. [1967], becoming the Group Method of Data Handling

1.2.1.2 A Description

Among the different networks existing, the artificial neural networks (ANNs) and the artificial recurrent neural networks (RNNs) are computational models designed by more or less detailed analogy with biological brain modules. The former is presented in Chapter (4) and the latter is introduced in Chapter (5). ANNs, formalised by McCulloch et al. [1943], are a class of generalised nonlinear nonparametric models inspired by studies of the human brain. They get their intelligence from learning process, giving them the capability of auto-adaptability, association, and memory to perform certain tasks. The backpropagation network, which is the most popular and the most widely implemented neural network in the financial industry, is based on a multi-layered feedforward topology with supervised learning (see Refenes et al. [1997]). The network is fully connected, with every node in the lower layer linked to every node in the next higher layer via weight values. The learning of the backpropagation neural network is based on an error minimisation procedure where the weights are modified according to an error function comparing the neural network output with the training targets. We evaluate the derivatives of the error function with respect to the weights which are used to compute the adjustment to be made to the weights. The simplest such techniques involves gradient descent. In the case of time series prediction, a set of input-target pairs is created, forming the training samples. Every time a

Electronic copy available at: https://ssrn.com/abstract=3307566

new time series is generated, new observations are added to the set and the oldest ones are dropped out. Alternatively, artificial recurrent neural networks (RNNs) are a class of feedback artificial neural network architecture that uses iterative loops to store information, which is inspired by the cyclical connectivity of neurons in the brain. The existence of cycles allows RNNs to develop a self-sustained temporal activation dynamics along its recurrent connection pathways, even in the absence of input, making them a dynamical system. This feature of the RNNs make them attractive for processing serially correlated time series. Unlike the ANNs and traditional time series models, the RNNs are flexible enough, and avoids taking the size of sliding windows into account because they can decide what to store and what to ignore during the learning process.

In real-time data, training neural networks involves the optimisation of non-convex objective functions, which makes the computation of the gradient descent challenging and in turn leads to a costly or even infeasible learning process. One solution is to randomly assign a subset of the networks' weights, so that the resulting optimisation task can be formulated as a linear least-squares problem. Numerous experimental results indicate that such randomised models can match, or even improve, adaptable ones, with a number of favourable benefits, such as, simplicity of implementation, faster learning with less intervention from human beings, possibility of leveraging over all linear regression and classification algorithms. Such improvements in standard RNNs design were proposed independently by Maass et al. [2002] under the name of Liquid State Machines and Jaeger [2001] under the name of Echo State Networks (ESN). Over time, these types of models became known as Reservoir Computing (RC). RC assumes that supervised adaptation of all inter-connection weights are not necessary, and only training a memoryless supervised readout from it is sufficient to obtain good results. Thus, it avoids the shortcomings of the gradient-descent training of RNNs. RC is based on the computational separation between a dynamic reservoir and a recurrence-free readout (see details in Section (5.3)).

1.2.2 Modelling changing environment

1.2.2.1 Multiple characteristic timescales

In general, the dynamics of RNN models evolve according to precise timing, while real-world time series are characterised by multiple seasonalities (or patterns). Substantial literature exists on modelling different temporal resolutions (multiple characteristic timescales), especially in presence of long-term relations, among which the Long Short Term Memory (LSTM) became a reference (see Hochreiter et al. [1997]). Several variants have been proposed to improve the simultaneous learning of long and short time relationships (see Cho et al. [2014]). Other methods developed based on the idea that temporal relationships are hierarchically structured, so that RNNs should be organised accordingly (see El Hihi et al. [1995]). It led some models to be implemented by stacking multiple recurrent layers (see Graves [2013], Chung et al. [2015]). In Reservoir Computing, characterised by fading memory preventing to model slow periodicities, solutions were proposed to slow down the dynamics of the reservoir by considering digital bandpass filters. The latter encourages the decoupling of the dynamics within a single reservoir by identifying particular frequency-domain characteristics. A large amount of filters are initialised with random cutoff frequencies to encode a wide range of timescales in the recurrent layer, providing the correct timings to capture the target dynamics. Alternatively, the filters can be manually tuned based on the information of the frequency spectrum of the desired response signal (see Siewert et al. [2007], Wyffels et al. [2008]).

An important development in machine learning was the use of ensemble models, which simultaneously combine different types of classifiers. The idea being that classifiers of different types and/or using different data exhibit distinct strengths and weaknesses. As a result, it has been assumed that combining multiple classifiers could improve prediction accuracy (see Ho et al. [1994]). Jaeger [2004] discussed the fact that a single reservoir could not train a multiple superimposed oscillator (MSO). We know from classic kernel-based signal representation (Fourier and wavelet transform) that an individual kernel can represent an almost unique portion of the target signal. Further, a single reservoir can reproduce the behaviour of a sine-wave generator (see Jaeger [2001]). We can therefore construct

Electronic copy available at: https://ssrn.com/abstract=3307566

an ESN with multiple reservoirs, each of which having a different configuration, to generate multiple sine waves of different frequencies. The problem being to define the grouping of neurons into several separate reservoir in a non-redundant manner.
Wiestra et al. [2005] explained that all the neurons in the same reservoir are coupled, when the task requires the simultaneous existence of multiple decoupled internal states. One solution was to consider an evolutionary-based reservoir design method, called Evolino. Xu et al. [2007] proposed the decoupled echo state network (DESN) involving the use of lateral inhibition. It is a structure making use of multiple reservoirs competing with each other through inhibitory connections and combining the internal states of all the reservoirs to form the output signal. Bianchi et al. [2017] proposed a network made of several recurrent groups of neurons trained to separately adapt to each timescale of the input signal. The recurrent layer of the network is randomly generated, but the connections must form a structure with K groups of bandpass neurons where each group $\mathcal{C}_k$ contains N neurons strongly connected to the other neurons in $\mathcal{C}_k$. Further, a small amount of connections between neurons belonging to different groups are also generated. The bandpass parameters are learned via backpropagation.

In most of these methods a large number of parameters must be learned in order to adapt the network to process the right time scales. It results in a long training time with the possibility of overfitting data. In general, these parameters are retrieved through an analysis in the frequency domain or by a priori knowledge of the input data.

1.2.2.2 Long range dependence

In general, memory or persistent internal states can not be modelled by fading memory systems (see Definition (8.1.1)). However, the latter can be enlarged through feedback from trained readouts. Maass et al. [2005a] [2005b] presented a computational theory characterising the gain in computational power achieved through feedback in dynamical systems with fading memory. As a result, anti-persistent systems using feedback can obtain universal computational capabilities similar to those of persistent systems. It leads to high-dimensional attractor-based models capable of capturing the characteristics of persistent signal response.

In the field of autonomous robot locomotion, stable pattern generation and robustness against perturbations is necessary. For instance, learning rhythmic stable motions in a controllable way is usually modelled with central patterns generators (CPG). In this framework, the output feedback can generate a suitable slow dynamics with high precision. As a result, RC models with output feedback were proposed to generate CPGs with interesting properties. In order to properly feedback the forecast and to modify appropriately the output weights if needed, one needs to implement an online algorithm. That is, when the reseroir is trained online, the model is adapted to recognising patterns and predicting chaotic time series (see Wyffels et al. [2009], Antonik et al. [2016]).

Reinhart et al. [2008] [2009] provided a new approach for training offline, or online, algorithms based on the transient/attractor concepts called Associative Reservoir Computing (ARC). As an example, an associative connection between two entities is bidirectional. The process denotes the recall of one entity from the other. Further, Reinhart [2011] introduced the concept of forward and inverse models in RC models. If the forward relation is many-to-one, the reverse relation becomes ambiguous, that is, one-to-many. Thus, there is a loss of information. One solution is to use additional information that is integrated over time. It can be modelled with an adaptive dynamical system to facilitate bidirectional and continuous association. The author pointed out the necessity of having a feedback in bidirectional association context. The main idea being to solve the ambiguous inverse problems by letting the model remain in an attractor network by teaching the reservoir states to stay stable. At every time step, the algorithm makes the entire network (input, reservoir and output) converge, thus eliminating transient effects. It is very important when one considers an output feedback matrix as it might add transient effect. Since feedback of erroneous outputs into the network can lead to error amplification, the concept of output feedback dynamics is formalised and an output feedback stability criterion is proposed.

Electronic copy available at: https://ssrn.com/abstract=3307566

1.2.2.3 The variable weight neural network

In conventional neural networks, connection weights are fixed. That is, the characteristics of the neural network is invariant (the invariant type neural network). The adaptive ability of neural networks to an unknown environment depends on its generalisation capability. However, there exists a limit of generalisation ability in invariant type neural networks. Yasuda et al. [2006] proposed the neural network with variable connection weights, which changes its connection weights and its characteristic according to the changing environment. Analysing an obstacle avoidance problem for an electric-powered wheelchair, they split the weights in two terms where the first term are basic quantities giving the basic motion of obstacle avoidance and the second terms adjust the connection weights in order to adapt the neural network to the change of environment in the vicinity of the wheelchair. The latter are calculated by using another neural network, whose inputs are outputs of PSD sensors. As a result, the law producing avoidance orders, changes according to the current condition of the obstacle in the vicinity of the wheelchair. Thus, this mechanism adapts the characteristics of neural network to the changing environment at every moment and corrects the generation algorithm of obstacle avoidance operation appropriately.
Lam et al. [2014] presented the variable weight neural network (VWNN), allowing its weights to be changed in operation according to the characteristic of the network inputs. Analysing the problem of surface material recognition and epilepsy seizure phases recognition, they demonstrated its ability to adapt to different characteristics of input data resulting in better performance than with fixed weights neural networks. A VWNN consists of two traditional neural networks, namely tuned and tuning neural networks. The former is the one which actually classifies the input data, while the latter provides the weights to the tuned neural network according to the characteristic of the input data. The VWNN works on the principle that the connection weights are function to the external input signal. Thus, in theory, the VWNN can be viewed as an infinite number of traditional neural networks with fixed weights.
A description is given in Appendix (5.4.1) and an application to financial time series is presented.

1.2.3 Interpreting neural networks

Neural networks (NNs) are based on predefined equations or formulae (see Section (8.1)) and have the capability of modelling complex statistical interactions between features for automatic feature learning. In order to perform a task, data scientists create an architecture (topology) whose (meta)-parameters are able (or not) to provide correct answers when some new input data is presented. These parameters are the key to their knowledge (see Palmer et al. [2002]), which depends on their generalisation ability.
Given this singular way of learning, NNs have been treated as black box models for their lack of interpretability compared with classical statistical models. That is, their inability to know in an explicit way the relations established between explanatory variables (input) and dependent variables (output).
Since the recent applications of NNs in the industry are intended to make critical decisions, it has become paramount to understand how these models make predictions. To do so, academics focused on explaining individual feature importance and determining the contribution of explanatory variables. One need first to discover (detect) the interactions and then interpret them, which is not an easy task. The methods used for detecting interactions from NNs are usually statistical methods. In general, they rely on the assumption that the response signal is stationary with independent and identically distributed increments. Further, these tests rely on the idea that NNs are capable of successfully modelling complex statistical interactions.

1.2.3.1 Explaining the input feature importance

Several approaches have been proposed to interpret NNs, classified as follows:

1. direct interpretation: explaining individual feature importance (Ross et al. [2017]), developing attention-based models, which illustrate where neural networks focus during inference, providing model-specific visualisations, such as feature map and gate activation visualisations (see Yosinski et al. [2015]).

Electronic copy available at: https://ssrn.com/abstract=3307566

2. indirect interpretation: post-hoc interpretation of feature importance (see Ribeiro et al. [2016]), knowledge distillation to simpler interpretable models.

The discovery of interactions (or statistical interaction detection) has become an essential part of the literature on neural networks. Two general approaches exist:

1. to conduct individual tests for each combination of features. Examples are ANOVA and Additive Groves (see Sorokina et al. [2008]). Both methods are computationally expensive in dimension higher than three.

2. to pre-specify all interaction forms of interest, then use Lasso to simultaneously select which are important. They are quick at selecting interactions but require specifying all interaction terms of interest (see Sun [1999]).

Interpreting NNs is far more difficult, but new methods have recently developed for traditional feedforward form and deep architectures. Methods such as feature map visualisation, de-convolution, saliency maps have been especially important to the vision community for understanding how convolutional networks represent images. In the case of long short-term memory networks (LSTMs), a research direction has studied multiplicative interactions in the unique gating equations of LSTMs to extract relationships between variables across a sequence (see Arras et al. [2017], Murdoch et al. [2018]). Tsang et al. [2018] proposed the Neural Interaction Detection (NID), which detects statistical interactions of any order or form captured by a feed-forward neural network, by examining its weight matrices. Top-K true interactions are determined from interaction rankings by using a special form of generalised additive model, which accounts for interactions of variable order.

1.2.3.2 Determining the contribution of explanatory variables

Several methods have been proposed to assess the relative importance (contribution) of each explanatory variable: methods from the family of sensitivity analyses (SA) (perturb method, profile method) (see Cortez et al. [2013]), numeric sensitivity analyses (NSA) computing the slope between input and output (partial derivative method (PaD)) and even the curvature effects (second order derivatives), and methods specific to the topology of NNs (connection weights method). In general, an optimal NN architecture is selected and several variable contribution methods are applied. However, when applying several methods to optimal or suboptimal NN architecture, the importance ranking of variables differs from method to method (high variability), indicating their inherent instability. As a result, there is no consensus on which model is the best for determining the contribution of variables. The difficulty of choosing a single optimal NN model led some authors to consider a set of good-performing NN models called neural network committee (NNC) (see Cao et al. [2008]). To mitigate these variabilities, De Ona et al. [2014] suggested to apply the methods on a set of NNs with the same architecture and trained with identical learning algorithm, activation functions, momentum value and learning ratio. The calculation of the average values of the relative importance of variables was used to determine the contribution of variables. The variables Frequency, Speed, Punctuality and Proximity are classified as the most important by all methods.

1.2.4 Optimisation for machine learning

1.2.4.1 Function estimation

In machine learning, given the samples (x_i, y_i), $i = 1, 2, ..., m$, of m pairs of input vector $x_i \in \mathbb{R}^{N_i}$ and labelled output vectors $y_i \in \mathbb{R}^{N_o}$, a task for supervised learning consists in learning a functional relation between the input x_i and the desired output y_i, for the training set (see Section (8.1)).
The learning can be understood as finding a mapping $h(\cdot, w)$ such that $h(x_i, w) \approx y_i$ with w an N_w dimensional vector of parameters to be learned. Thus, the goal of machine learning is to minimise the expected loss (also called risk function)

$$\widehat{L}(h) = E[L(h(X), Y)]$$

Electronic copy available at: https://ssrn.com/abstract=3307566

However, we do not know the joint probability distribution $P(X,Y)$ and can not estimate it. One solution is to consider Empirical Risk minimisation:

- we substitute the sample mean for the expectation
- minimise empirical loss:

$$\overline{L}(h) = \frac{1}{m}\sum_{i=1}^{m} L(h(x_i), y_i)$$

  Also known as (A.K.A.) sample average approximation (see Appendix (14.5)).

Mathematically, we want to find

$$w^* = \arg\min_w \overline{L}(w)$$

where $\overline{L}(w) = \frac{1}{m}\sum_{i=1}^{m} L(w; x_i, y_i))$ approximates the expected value of the loss. We can also write the loss function as $L(h(x_i, w), y_i)$.
In general, the relation between the input x_i and the desired output y_i can either be solved by a linear model as

$$\widehat{y}_i = W^\top x_i$$

where $W \in \mathbb{R}^{N_i \times N_o}$ is a weight matrix, or by a nonlinear model

$$\widehat{y}_i = f(W^\top x_i)$$

where $f(\cdot)$ is the activation function.
The training of neural networks (NNs) depends on the adaptation of free network parameters, the weight values. Thus, it is an optimisation task where the result is to find optimal weight set of the network that reduce an error function E. It can be formulated as the minimisation of an error function in the space of connection weights (see Chapter (10)). In the case of our loss function the minimisation problem is

$$\min_W \widehat{L}(h, W)$$

In practice, the function f in the above minimisation problem is not given explicitly but only implicitly through some examples. While convex problems are theoretically and algorithmically much more tractable, the complexity of non-convex problems can grow enormously (NP-hard). Global optimisation addresses the problems of non-convex optimisation.

1.2.4.2 Parameters estimation

The methods for training neural networks (NNs) are backpropagation (BP), Levenberg-Marquadt (LM), Quasi-Newton (QN). In general, the error backpropagation method (EBP), based on gradient method, is preferred. However, the objective function describing the neural networks training problem is a multi-modal function, so that the algorithms based on gradient methods can easily be stuck in local extremes. To avoid this problem one can consider using a global search optimisation (see Section (12.1)).

Over time, the relationship between mathematical programming models (MP) and machine learning models have been increasingly coupled. Methods developed for introducing constraints into the learning model in view of adding domain knowledge to enforce non-negativity and sparsity in dimensionality reduction methods. Further, methods were developed for solving existing machine learning models more efficiently. As data set size grows, MP models became inadequate and methods were developed to exploit the properties of learning problems in machine learning. In turn, machine learning has motivated advances in mathematical programming: the optimisation problems arising from

Electronic copy available at: https://ssrn.com/abstract=3307566

large scale machine learning and data mining far exceed the size of the problem typically reported in the mathematical programming literature.

Generally, an ANN model is specified by its topology, node characteristics and the training algorithms. There are a variety of ANNs with various topologies, node properties and training algorithms. See for instance Section (5.3) in the case of Reservoir Computing. These characteristics (or features) must be considered when designing a neural network model, leading to large families of models. The data scientist must select an appropriate family of models and massages the data into a format amenable to modelling. Then the model is typically trained (Training) by solving a core optimisation problem that optimises the variables (or parameters) of the model with respect to the selected loss function and possibly some regularisation function. The efficiency of an optimisation method often depends on the choosing of a number of behavioural parameters (meta parameters). In that case, the mapping becomes

$$h(x_i, w, \theta) \approx y_i$$

with θ an N_θ dimensional vector of parameters to be learned. Hence, in the process of model selection and validation (), the core optimisation problem may be solved many times. It led researchers to combine mathematical programming with machine learning. The interaction of state-of-the-art machine learning and mathematical programming has been called Large Scale Optimization and Machine Learning leading to improved machine learning models as well as new uses of mathematical programming in machine learning.

One solution consists in performing an additional optimisation aiming at minimising the RMSE on these meta parameters Θ. It can be done on the validation sample. In that case, the minimisation problem is

$$\min_{\Theta} \widehat{L}(h, W^*, \Theta)$$

Pedersen et al. [2008b] proposed an overlaying optimisation method known as meta-optimisation. The idea is to have an optimisation method act as an overlaying meta-optimiser, trying to find the best performing behavioural parameters for another optimisation method (see Pedersen et al. [2008a]).

1.3 The learning process

1.3.0.1 Prediction tasks

The strength of neural networks (NNs) compared to other techniques is their high capacity for classification, prediction and failure tolerance. In financial markets, NNs are generally applied for prediction tasks and for classification (see Section (2.2)). It can be summarised as follows:

1+ Chaotic series prediction: performing time series forecast.
In the academic literature, chaotic systems are generally described by Mackey-Glass equations (see Mackey et al. [1977]). It is generated from the following non-linear time delay differential equation:

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x^n(t-\tau)} - bx(t)$$

where a, b, n and τ are real numbers. τ is the delay parameter. Depending on the parameter set we choose, the Mackey-Glass signal exhibits various characteristics. Figure (1.1) below demonstrates a particular behaviour, which is at the same time very chaotic. Similarly to a financial time series we observe: an identified behaviour (trending or mean-reverting) together with an unidentified one (chaos). Examples are next period price change and next period actual price (opening, high, low and closing prices), which concerns the change in price with respect to the next period and the numerical value for the actual next period price, respectively.

Electronic copy available at: https://ssrn.com/abstract=3307566

2+ Autonomous pattern generation: classification problem.
A pattern is a short sequence of randomly chosen real numbers that is repeated periodically to form an infinite time series. In general, the length of the sequence may be set from 1 up to the number of neurons N. In finance patterns are distinctive formations of market prices. For example, the next period direction concerns the direction of the price movement with respect to the next period. In that case we classify each record as Up or Down. The results are generally obtained by using only historic prices and technical indicators (TIs), which are mathematical transformations of market returns. The most common TIs are the simple moving average (SMA), exponential moving average (EMA), relative strength index (RSI), rate of change (ROC), moving average convergence / divergence (MACD), William's oscillator and average true range (ATR).

In general, academics and practitioners focus on case $(1+)$, which implies making a statement about the true dynamics (denoted f) followed by market returns. While in case $(2+)$ we only consider part of the probability space, this approach seems ignored by academics. From Computational Learning, an approximation model (the hypothesis, denoted h) must be devised and trained on the full measurement space (case $(1+)$) or part of the space (case $(2+)$). In order to assess if the function h is approximately correct, a measure of error must be considered (the cost function). This process requires sophisticated search and examination methods (optimisation algorithms) such as machine learning.

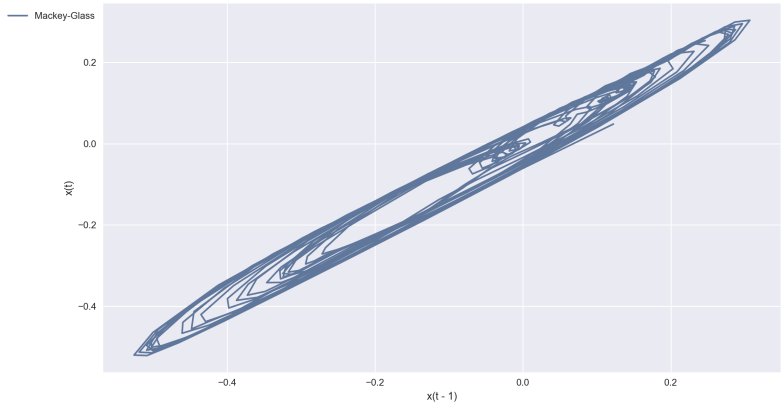


Figure 1.1: Mackey-Glass Behaviour for (a, b, n, τ, x_0) = (0.2, 0.01, 10, 17, $random$)

1.3.0.2 Training and exploitation of the model

When forecasting time series or, generating autonomous patterns, we consider the following three steps:

1. Training: it consists in determining the model parameters that minimise the cost function.
2. Optimisation, and
3. Adaptability.

We split the available data into 3 samples: Training, Validation and Testing. It is called sample hashing (see an example in Figure (1.2)). Optimisation is performed in the validation set. Testing is the sample where the calibrated model is assessed for generalisation. That is, the model is ran without any exhaustive filter nor complementary optimisation. The learning process is as follows:

Electronic copy available at: https://ssrn.com/abstract=3307566

- In the case of pattern generation: During the training phase, the reservoir computer receives the pattern signal as input and is trained to predict the next value of the pattern from the current one. The length of the training input sequence is measured in terms of the pattern length, that is, how many times the whole pattern is presented to the reservoir computer. The duration of the training process depends on the length of the pattern. For instance, short patterns are successfully learnt after 100 repetitions, while long patterns need to be shown up to $50k$ times. The RMSE is used to evaluate the training phase. After the training phase, the reservoir input is switched from the training sequence to the reservoir output signal, and the system is left running autonomously.
- In the case of series prediction, training and test phases are the same as for pattern generation. During training, the reservoir receives the time series and is trained to perform a one-step ahead prediction, while in the test phase the reservoir receives its own output and the system runs autonomously.

Based on the classification task chosen, several possibilities exist for training and exploiting models:

1* training on transient process → exploitation on transient process

2* training on attractor-based model → exploitation on attractor-based

3* training on attractor-based model → exploitation on transient process

4* training on attractor-based model → exploitation on transient process

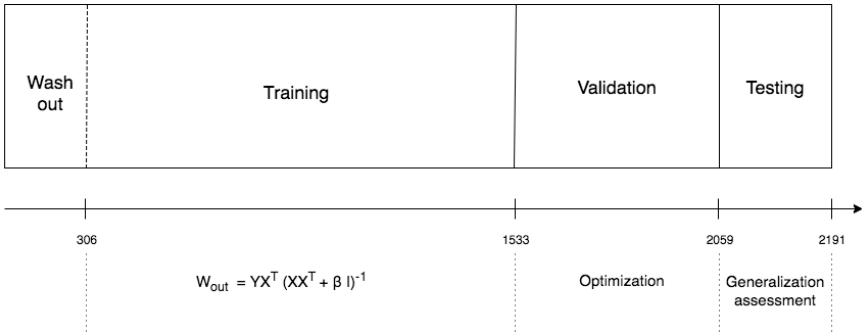


Figure 1.2: Sample hashing: Training sample is $washed - out$ at 20% meaning that the Ridge regression is only performed on the 80% remaining. By doing this, we get rid of the reservoir initial transient.

1.3.0.3 The framework

1.3.0.3.1 The input variables When selecting the right input variables, Krollner et al. [2010] showed that academics and practitioners rely on some form on lagged index data (case $(1+)$ in Section (1.3)) rather than considering autonomous pattern generation (case $(2+)$). Examples of articles that focus on forecasting market returns are Zeng et al. [2006], Zhang et al. [2007], Zhu et al. [2008], Lu et al. [2009], Liao et al. [2010]. Based on some well known techniques combining forecasts (see Armstrong [2001]), several studies showed that combining forecasts (using NN models) could greatly enhance the forecast effectiveness (see Wang et al. [2008], Chitra et al. [2010]).
Alternatively, traders and academics have used technical analysis as well as certain fundamental valuation metrics to improve results (see Gencay [1997], Metghalchi et al. [2007]). Further, some statisticians showed that technical indicators have predictive power and found a good measure of support for their effectiveness (see Chen et al. [2009], Metghalchi et al. [2012] [2015]). As a result, these strategies have been incorporated in machine learning models to forecast future trend direction (case $(2+)$). The results obtained are achieved using only historic prices and technical indicators (see Kim rt al. [2006], Bekiros et al. [2008], Kara et al. [2011], Dingli et al. [2017]). Some authors also used technical indicators to forecast both future price levels and asset price directions (see Patel et al. [2015]).

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, one can directly use continuous technical indicators (actual time series) or one can choose to represent them as trend deterministic data (discrete in nature). Van den Poel et al. [2016] combined in some way continuous technical indicators, technical trading rules and past returns to become input to predictor models. They proposed a predictive model to predict future stock price direction accurately.

1.3.0.3.2 The data The different forecasting outputs are computed for the following periods: daily, weekly, monthly, quarterly and yearly. However, an increasing number of articles consider intra-day high frequency data, which requires highly liquid markets. In that case the data comprises the date and time, open-high-low-close price levels and the volume traded for each underlying asset. One of the main reason for considering intraday trading horizon is the notion of self-destruction of predictable patterns in stock prices (see Timmermann et al. [2004]). For the data to be consistent, historical prices and/or volume are usually adjusted to reflect spin-offs, stock splits/consolidations, stock dividend/bonus and right offerings/entitlement.

1.3.0.3.3 Evaluation methods Most evaluation methods of the ML models are benchmark models where the proposed improved version is compared against the original one using a forecast error as an evaluation metric. Examples of benchmark models are buy and hold, random walk, statistical techniques and other machine learning techniques.

1.3.0.3.4 The measures Linear measures are measurements in one direction. For example length, width, or height of objects, and distance between two points are linear measures. Linearity also also express the idea that the model possesses the property of additivity and homogeneity, meaning that a change in one variable causes a proportional change in another variable. An example of a linear measure is the root-mean-square deviation (RMSD) or root-mean-square error (RMSE), which is used to measure the differences between values predicted by a model or an estimator and the values observed.
In mathematics and science, a nonlinear system is a system in which the change of the output is not proportional to the change of the input. Systems can be defined as nonlinear, regardless of whether known linear functions appear in the equations. Nonlinearity measures are a means of quantifying the size of nonlinearity in the Input/Output-behaviour of nonlinear systems (more than one dimension). Methods that efficiently compute nonlinearity measures are based on convex optimisation.
Note, linear measures do not account for the non-linearity of RNN models (see Bellgard et al. [1999]). One way around is to assess the performance of a model using trading simulation.

1.3.1 Using neural networks to model returns

1.3.1.1 An overview

Among the various techniques to classifying and forecasting financial time series, fundamental and technical analysis are the most popular ones. Even though statistical procedures are widely used for patterns recognition, the effectiveness of these methods relies both on model's assumptions and prior knowledge on data properties. To remedy these pitfalls, several classifiers developed, using various data mining and computational intelligence methods such as rule induction, fuzzy rule induction, decision trees, neural networks etc. In 1990, Kimoto et al. [1990] applied a modular neural network machine learning algorithm to predict the movement of stock index of Tokyo Stock Exchange. Since ANN developed further, it has been widely applied to stock analysis. Nowadays, the best recognised tools in the currency markets is the artificial neural networks (ANNs), supported by numerous empirical studies (see Ahmed et al. [2010]). Similarly, Krollner et al. [2010] provided a survey on forecasting financial time series with machine learning and found that ANNs were the dominant technique in that field. The foremost reason for using ANNs is that there is some nonlinear aspect to the forecasting problem under consideration, taking the form of a complex nonlinear relationship between the independent and dependent variables. The characteristics of financial time series, such as equity stock or currency markets, are influenced by the psychology of traders (behavioural finance) and are strongly non-linear and hardly predictable (see Maknickiene et al. [2011]).

Electronic copy available at: https://ssrn.com/abstract=3307566

One advantage in using an ANNs is that the researcher does not need to know a priori the type of functional relationship existing between the independent and dependent variables (see Darbellay et al. [2000]). A vast literature demonstrated that neural network performs better than conventional statistic approaches in financial forecasting (see Refenes et al. [1994], Adya et al. [1998], Abu-Mostafa et al. [2001]). For many financial forecasting problems, classification models work better than point prediction. Further, Qi [2001] argued that due to the continually changing nature of financial relationships, ANNs are more likely to outperform traditional techniques when the input data is kept as current as possible. This is done by recursive modeling, where the researcher adds new observations and drops the oldest ones each time a new time series forecast is made (sliding window). Olson et al. [2003] compared neural network forecasts of one-year ahead Canadian stock returns with the forecasts obtained by using ordinary least squares (OLS) and logistic regression techniques and showed that the backpropagation algorithm outperformed the best regression alternatives for both point estimation and in classifying firms expected to have either high or low returns This superiority of the NNs translated into greater profitability using various trading rules. Using data from four major stock market indexes, Fok et al. [2008] compared linear regression and neural network backpropagation by testing their forecasting performances. They showed that the latter had better prediction accuracy.

1.3.1.2 Combining neural networks with evolutionary algorithms

Some of the disadvantages of NNs are long training time, unwanted convergence to local instead of global optimal solution, and large number of parameters (see Lee et al. [1991]). One way forward is to combine ANN with another algorithm such as evolutionary computing tools that can take care of a specific problem (see Section (12.1)). Many research papers have appeared in the literature using evolutionary computing tools, such as genetic algorithm (GA) (see Montana et al. [1989], Nair et al. [2011], Contras et al. [2016]), particle swarm optimization (PSO) (see Kennedy et al. [1995] Jordehi et al. [2013]), bacterial foraging optimization (BFO) (see Jhankal et al. [2011]), and Adaptive bacterial foraging optimization (ABFO) in developing forecasting models.
Inspired by the pattern exhibited by bacterial foraging behaviour, Helstrom et al. [1998] proposed an evolutionary computation technique called Bacterial foraging optimization (BFO). It analyses how the run-length unit parameter of BFO controls the exploration of the whole search space and the exploitation of the promising areas. Shin et al. [1998] proposed data mining approach using genetic algorithms (GA) to solve the knowledge acquisition problems that are inherent in constructing and maintaining rule-based applications for stock market. Kim et al. [2000] used GA to improve the learning algorithm, and also to reduce the complexity of the feature space. Kim et al. [2004b] developed a feature transformation method using genetic algorithms. It reduces the dimensionality of the feature space and removes irrelevant factors involved in stock price prediction. Jamous et al. [2016] considered particle swarm optimization with center mass (PSOCOM) technique to develop an efficient forecasting model. Rani et al. [2017] presented an overview on optimisation techniques and hybridisation to improve ANN performance (see details in Section (12.1)).

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 2

Data, information and knowledge

2.1 Introduction to information theory

See textbooks by Shannon et al. [1949], MacKay [2003], among others.

2.1.1 Presenting a few concepts

2.1.1.1 Uncertainty

According to Shannon, information is any entity or form that provides the answer to a question of some kind, or resolves uncertainty. It is thus related to data and knowledge, as data represents values attributed to parameters, and knowledge signifies understanding of real things or abstract concepts. In the case of data, the existence of information is not necessarily coupled to an observer (for example, it exists beyond an event horizon), while in the case of knowledge, the information requires a cognitive observer.
Information is conveyed either as the content of a message or through direct or indirect observation. That which is perceived can be construed as a message in its own right, and in that sense, information is always conveyed as the content of a message. Information can be encoded into various forms for transmission and interpretation (for example, information may be encoded into a sequence of signs, or transmitted via a signal). It can also be encrypted for safe storage and communication.
Shannon proposed that information reduces uncertainty and therefore reduces entropy. A typical example of this principle is flipping a two-sided coin. When tossing a coin, we are uncertain about the two possible outcomes. By observing the outcome (say, heads), we gain information that reduces the uncertainty about this event. The uncertainty of an event is measured by its probability of occurrence and is inversely proportional to that. The more uncertain an event, the more information is required to resolve uncertainty of that event. Shannon built on the work of fellow researchers Ralph Hartley and Harry Nyquist to reveal that coding and symbols were the key to resolving whether two sides of a communication had a common understanding of the uncertainty being resolved. Following the example of the coin he concluded that any type of information, could be encoded as a series of fundamental yes or no answers. Nowadays this is known as bits of digital information, ones and zeroes, that represent everything from email text, digital photos, compact disc music or high definition video. The concept of information became closely related to notions of constraint, communication, control, data, form, education, knowledge, meaning, understanding, mental stimuli, pattern, perception, representation, and entropy.

2.1.1.2 Coding

Information theory (IT) studies the transmission, processing, extraction, and utilisation of information. Abstractly, information can be thought of as the resolution of uncertainty. In the case of communication of information over a

Electronic copy available at: https://ssrn.com/abstract=3307566

noisy channel, this abstract concept was made concrete by Shannon [1948], in which "information" is thought of as a set of possible messages, where the goal is to send these messages over a noisy channel, and then to have the receiver reconstruct the message with low probability of error, in spite of the channel noise. Shannon introduced information theory (IT) by proposing some fundamental limits to the representation and transmission of information. In the noisy-channel coding theorem, he showed that, in the limit of many channel uses, the rate of information that is asymptotically achievable is equal to the channel capacity, a quantity dependent merely on the statistics of the channel over which the messages are sent.
Since then, information theory (IT) is a broad and deep mathematical theory, with equally broad and deep applications, amongst which is the vital field of coding theory. It has provided the theoretical motivations for many of the outstanding advances in digital communications and digital storage. Coding theory is concerned with finding explicit methods, called codes, for increasing the efficiency and reducing the error rate of data communication over noisy channels to near the channel capacity. These codes can be roughly subdivided into data compression (source coding) and error-correction (channel coding) techniques.
One generally uses the following digital communication model in the transfer of information

1. The source is the source of (digital) data
2. The source encoder serves the purpose of removing as much redundancy as possible from the data. It is called the data compression portion.
3. The channel coder puts a modest amount of redundancy back in order to perform error detection or correction.
4. The channel is what the data passes through, possibly becoming corrupted along the way.
5. The channel decoder performs error correction or detection.
6. The source decoder undoes what is necessary to recover the data back.

There are other blocks that could be inserted in that model, such as

- a block enforcing channel constraints sometime called a line coder.
- a block performing encryption/decryption.
- a block performing lossy compression.

2.1.1.3 Randomness

One of the key concept in IT is that information is conveyed by randomness, that is, information is defined in some mathematical sense, which is not identical to the one used by humans. However, it is not too difficult to make the connection between randomness and information. Another important concept in IT is that of typical sequences. In a sequence of bits of length n, there are some sequences which are typical. For example, in a sequence of coin-tossing outcomes for a fair coin, we would expect the number of heads and tails to be approximately equal. A sequence not following this trend is thus atypical. A good part of IT is capturing this concept of typicality as precisely as possible and using it to concluding how many bits are needed to represent sequence of data. The main idea being to use bits to represent only the typical sequences, since the other ones are rare events. This concept of typical sequences corresponds to the asymptotic equipartition property.
Shannon for the first time introduced the qualitative and quantitative model of communication as a statistical process underlying information theory. That is, information theory is based on probability theory and statistics. It concerns itself with measures of information of the distributions associated with random variables. Important quantities of information are entropy, a measure of information in a single random variable, and mutual information, a measure of information in common between two random variables. Given a discrete random variable X, with x a particular

Electronic copy available at: https://ssrn.com/abstract=3307566

outcome occurring with probability $p(x)$. Then, we assign to that event x the information that it conveys via the uncertainty measure

$$\text{uncertainty } = -\log p(x)$$

where the base of the logarithms determines the units of information. The units of $\log_2$ are in bits, that of $\log_e$ are in nats. For example, a random variable having two outcomes, 0 and 1, occurring with probability $p(0) = p(1) = \frac{1}{2}$, then each outcome conveys 1 bit of information. However, if $p(0) = 1$ and $p(1) = 0$, then the information conveyed when 0 happens is 0. We get no information from it, as we knew all along that it would happen. Hence, the information that 1 happens is ∞, as we are totally surprised by its occurrence. In general, one commonly use the average uncertainty provided by a random variable X taking value in a space χ, leading to the notion of entropy.
Much of the mathematics behind information theory with events of different probabilities were developed for the field of thermodynamics by Ludwig Boltzmann and J. Willard Gibbs. Connections between information-theoretic entropy and thermodynamic entropy, including the important contributions by Landauer in the 1960s (see Landauer [1961]), are explored in Entropy in thermodynamics and information theory.

2.1.2 Some facts on entropy in information theory

The entropy functional, defined on a Markov diffusion process, plays an important roll in the theory of information and statistical physics, informational macrodynamics and control systems. In information theory (IT), entropy is the average amount of information contained in each message received. That is, entropy is a measure of unpredictability of information content. Named after Boltzmann's H-theorem, Shannon [1948] defined the entropy H of a discrete random variable X with possible values $\{x_1, .., x_n\}$ and probability mass function $P(X)$ as

$$H(X) = E[I(X)] = E[-\log P(X)]$$

where $I(X)$ is the information content of X, which is itself a random variable. When taken from a finite sample, the entropy can explicitly be written as

Definition 2.1.1 *The entropy $H(X)$ of a discrete random variable X taking values $\{x_1, .., x_n\}$ is*

$$H(X) = \sum_{i=1}^{n} p(x_i)I(x_i) = -\sum_{i=1}^{n} p(x_i)\log_b p(x_i)$$

where b is the base of the logarithm used.

The unit of entropy is bit for $b = 2$, nat for $b = e$ (where e is Euler's number), and dit (or digit) for $b = 10$. Given the well-known limit $\lim_{p\to 0_+} p\log p = 0$, in the case where $P(x_i) = 0$ for some i, the value of the corresponding summand is taken to be 0. For example, taking a fair coin, we get

$$H(X) = -(\frac{1}{2}\log\frac{1}{2} + \frac{1}{2}\log\frac{1}{2}) = 1 \text{ bit}$$

For a biased coin with $p(0) = 0.9$, we get

$$H(X) = -(0.9\log 0.9 + 0.1\log 0.1) = 0.469 \text{ bit}$$

If X is a binary random variable

$$X = \begin{cases} 1 \text{ with probability } p \\ 0 \text{ with probability } 1-p \end{cases}$$

then the entropy of X is

Electronic copy available at: https://ssrn.com/abstract=3307566

$$H(X) = -p\log p - (1-p)\log(1-p)$$

For some function $g(X)$ of a random variable, we get $E[g(X)] = \sum_{x\in\mathcal{X}} g(x)p(x)$ so that for $g(x) = \log\frac{1}{p(x)}$ we get

$$H(X) = E[g(X)] = E[\log\frac{1}{p(X)}]$$

We are now interested in the entropy of pairs of random variables (X, Y).

Definition 2.1.2 *If X and Y are jointly distributed according to $p(X, Y)$, then the joint entropy $H(X, Y)$ is*

$$H(X,Y) = -E[\log p(X,Y)] = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x,y)$$

Definition 2.1.3 *If $(X, Y) \sim p(x, y)$, the conditional entropy of two events X and Y is given by*

$$H(Y|X) = -E_{p(x,y)}[\log p(Y|X)] = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y|x)$$

This quantity should be understood as the amount of randomness in the random variable Y given that you know the value of X. The conditional entropy can also be written as

$$\begin{aligned} E[Y|X] &= -\sum_{x\in\mathcal{X}} p(x)\sum_{y\in\mathcal{Y}} p(y|x)\log p(y|x) \\ &= -\sum_{x\in\mathcal{X}} p(x)H(Y|X=x) \end{aligned}$$

Theorem 2.1.1 *chain rule*

$$H(X,Y) = H(X) + H(Y|X)$$

We can also have a joint entropy with a conditioning on it.

***Corollary* 1**

$$H(X,Y|Z) = H(X|Z) + H(Y|X,Z)$$

The inspiration for adopting the word entropy in information theory came from the close resemblance between Shannon's formula and very similar known formulae from statistical mechanics. In statistical thermodynamics the most general formula for the thermodynamic entropy S of a thermodynamic system is the Gibbs entropy

$$S = -k_B\sum_i p_i \ln p_i$$

where k_B is the Boltzmann constant, and p_i is the probability of a microstate. The Gibbs entropy translates over almost unchanged into the world of quantum physics to give the von Neumann entropy, introduced by John von Neumann in 1927

$$S = -k_B Tr(\rho\ln\rho)$$

where ρ is the density matrix of the quantum mechanical system and $Tr()$ is the trace. Note, at a multidisciplinary level, connections can be made between thermodynamic and informational entropy. In the view of Jaynes (1957), thermodynamic entropy, as explained by statistical mechanics, should be seen as an application of Shannon's information

Electronic copy available at: https://ssrn.com/abstract=3307566

theory: the thermodynamic entropy is interpreted as being proportional to the amount of further Shannon information needed to define the detailed microscopic state of the system, that remains uncommunicated by a description solely in terms of the macroscopic variables of classical thermodynamics, with the constant of proportionality being just the Boltzmann constant.

2.1.3 Relative entropy and mutual information

Another useful measure of entropy that works equally well in the discrete and the continuous case is the relative entropy of a distribution. Given a random variable with true distribution p, which we do not know due to incomplete information, instead we assume the distribution q. Then, the code will need more bits to represent the random variable, and the difference in bits is denoted by $D(p||q)$. The relative entropy is defined as the Kullback-Leibler divergence from the distribution to the reference measure q as follows.

Definition 2.1.4 *The relative entropy or Kullback-Leibler distance between two probability mass functions $p(x)$ and $q(x)$ is defined as*

$$D_{KL}(p||q) = E_p[\log \frac{p(x)}{q(x)}] = \sum_{x\in\mathcal{X}} p(x) \log \frac{p(x)}{q(x)}$$

This form is not symmetric, and q appears only in the denominator. Alternatively, assume that a probability distribution p is absolutely continuous with respect to a measure q, that is, it is of the form $p(dx) = f(x)q(dx)$ for some non-negative q-integrable function f with q-integral 1, then the relative entropy can be defined as

$$D_{KL}(p||q) = \int \log f(x)p(dx) = \int f(x) \log f(x)q(dx)$$

In this form the relative entropy generalises (up to change in sign) both the discrete entropy, where the measure q is the counting measure, and the differential entropy, where the measure q is the Lebesgue measure. If the measure q is itself a probability distribution, the relative entropy is non-negative, and zero if $p = q$ as measures. It is defined for any measure space, hence coordinate independent and invariant under co-ordinate reparametrisations if one properly takes into account the transformation of the measure q. The relative entropy, and implicitly entropy and differential entropy, do depend on the reference measure q.

Another important concept, called mutual information, describes the amount of information a random variable tells about another one. That is, observing the output of a channel, we want to know what information was sent. The channel coding theorem is a statement about mutual information.

Definition 2.1.5 *Let X and Y be random variables with joint distribution $p(X,Y)$ and marginal distributions $p(x)$ and $p(y)$. The mutual information $I(X;Y)$ is the relative entropy between the joint distribution and the product distribution*

$$\begin{aligned} I(X;Y) &= D(p(x,y)||p(x)p(y)) \\ &= \sum_{x\in\mathcal{X}} p(x) \sum_{y\in\mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \end{aligned}$$

Note, when X and Y are independent, $p(x,y) = p(x)p(y)$, and we get $I(X;Y) = 0$. That is, in case of independence, Y, can not tell us anything about X. An important interpretation of mutual information comes from the following theorem.

Theorem 2.1.2

$$I(X;Y) = H(X) - H(X|Y)$$

Electronic copy available at: https://ssrn.com/abstract=3307566

which states that the information that Y tells us about X is the reduction in uncertainty about X due to the knowledge of Y. By symmetry we get

$$I(X;Y) = H(Y) - H(Y|X) = I(Y;X)$$

Using $H(X,Y) = H(X) + H(Y|X)$, we get

$$I(X;Y) = H(X) + H(Y) - H(X,Y)$$

The information that X tells about Y is the uncertainty in X plus the uncertainty about Y minus the uncertainty in both X and Y. We can summarise statements about entropy as follows

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) \\ I(X;Y) &= H(Y) - H(Y|X) \\ I(X;Y) &= H(X) + H(Y) - H(X,Y) \\ I(X;Y) &= I(Y;X) \\ I(X;X) &= H(X) \end{aligned}$$

When dealing with the sequence of random variables $X_1, .., X_n$ drawn from the joint distribution $p(x_1, .., x_n)$, a variety of chain rules have been developed.

Theorem 2.1.3 *The joint entropy of $X_1, .., X_n$ is*

$$H(X_1, .., X_n) = \sum_{i=1}^{n} H(X_i|X_{i-1}, .., X_1)$$

The chain rule for entropy leads us to a chain rule for mutual information.

Theorem 2.1.4

$$I(X_1, .., X_n; Y) = \sum_{i=1}^{n} I(X_i; Y|X_{i-1}, ..., X_1)$$

2.1.4 Bounding performance measures

A large part of information theory consists in finding bounds on certain performance measures. One of the most important inequality used in IT is the Jensen's inequality. We are interested in convex functions because it is known that over the interval of convexity, there is only one minimum (for details see Appendix (14.3)). The following theorem describe the information inequality.

Theorem 2.1.5 $\log x \leqslant x - 1$, *with equality if and only if* $x = 1$.

We can now characterise some of the information measures defined above.

Theorem 2.1.6 $D(p||q) \geqslant 0$, *with equality if and only if* $p(x) = q(x)$ *for all* x.

***Corollary* 2** *Mutual information is positive,* $I(X;Y) \geqslant 0$, *with equality if and only* if X *and* Y *are independent.*

We let the random variable X takes values in the set $\mathcal{X}$, and we let $|\mathcal{X}|$ denotes the number of elements in that set. For discrete random variables, the uniform distribution over the range $\mathcal{X}$ has the maximum entropy.

Theorem 2.1.7 $H(X) \leqslant \log |\mathcal{X}|$, *with equality if and only if* X *has a uniform distribution.*

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, if you can show that some performance criterion is upper-bounded by some function, then by showing how to achieve that upper bound, we have found an optimum. We see that the more we know, the less uncertainty there is.

Theorem 2.1.8 *Condition reduces entropy:*

$$H(X|Y) \leqslant H(X)$$

with equality if and only if X and Y are independent.

Theorem 2.1.9

$$H(X_1, ..., X_n) \leqslant \sum_{i=1}^{n} H(X_i)$$

with equality if and only if X_i are independent.

The following theorem allow us to deduce the concavity (or convexity) of many useful functions.

Theorem 2.1.10 *Log-sum inequality*
For non-negative numbers $a_1, ..., a_n$ and $b_1, .., b_n$,

$$\sum_{i=1}^{n} a_i \log \frac{a_i}{b_i} \geqslant \big(\sum_{i=1}^{n} a_i\big) \log \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} b_i}$$

with equality if and only if $\frac{a_i}{b_i} = \ constant$.

The conditions on this theorem are much weaker than the one for Jensen's inequality, since it is not necessary to have the sets of numbers add up to 1. Using this inequality, one can prove a convexity statement about the relative entropy function.

Theorem 2.1.11 *If (p_1, q_1) and (p_2, q_2) are pairs of probability mass function, then*

$$D(\lambda p_1 + (1-\lambda)p_2||\lambda q_1 + (1-\lambda q_2)) \leqslant \lambda D(p_1||q_1) + (1-\lambda)D(p_2||q_2)$$

for all $0 \leqslant \lambda \leqslant 1$. That is, $D(p||q)$ is convex in the pair (p, q).

Further,

Theorem 2.1.12 *$H(p)$ is a concave function of p.*

Theorem 2.1.13 *Let $(X, Y) \sim p(x, y) = p(x)p(y|x)$. The mutual information $I(X; Y)$ is a concave function of $p(x)$ for fixed $p(y|x)$ and a convex function of $p(y|x)$ for fixed $p(x)$.*

The data processing inequality states that no matter what processing we perform on some data, we can not get more information out of a set of data than was there to begin with. In a sense, it provides a bound on how much can be accomplished with signal processing.

Definition 2.1.6 *Random variable X, Y, and Z are said to form a Markov chain in that order, denoted by $X \rightarrow Y \rightarrow Z$ if the conditional distribution of Z depends only on Y and is independent of X. (That is, if we know Y, then knowing X also does not tell us any more than if we only know Y). if X, Y, and Z form a Markov chain, then the joint distribution can be written*

$$p(x, y, z) = p(x)p(y|x)p(z|y)$$

Electronic copy available at: https://ssrn.com/abstract=3307566

The concept of a state is that knowing the present state, the future of the system is independent of the past. The conditional independence idea means

$$p(x,z|y) = \frac{p(x,y,z)}{p(y)} = \frac{p(x,y)p(z|y)}{p(y)} = p(x|y)p(z|y)$$

Note, if $Z = f(Y)$ then $X \to Y \to Z$.

Theorem 2.1.14 *Data processing inequality*
If $X \to Y \to Z$, then

$$I(X;Y) \geqslant I(X;Z)$$

If we think of Z as being the result of some processing done on the data Y, that is, $Z = f(Y)$ for some deterministic or random function, then there is no function that can increase the amount of information that Y tells about X.

2.1.5 Feature selection

A fundamental problem of machine learning is to approximate the functional relationship $f(\cdot)$ between an input $X = \{x_1, x_2, .., x_M\}$ and an output Y, based on a memory of data points, $\{X_i, Y_i\}$, $i = 1, .., N$, where the X_i are vectors of reals and the Y_i are real numbers. There are some cases where the output Y is not determined by the complete set of the input features $X = \{x_1, x_2, .., x_M\}$, but it is only decided by a subset of them $\{x_{(1)}, x_{(2)}, .., x_{(m)}\}$, where $m < M$. When we have sufficient data and time, we can use all the input features, including the irrelevant ones, to approximate the underlying function between the input and the output. However, in practice the irrelevant features raise two problems in the learning process.

1. The irrelevant input features will induce greater computational cost.
2. The irrelevant input features may lead to overfitting

As a result, it is reasonable and important to ignore those input features with little effect on the output, so as to keep the size of the approximator model small. While the feature selection problem has been studied by the statistics and machine learning communities for decades, it has received much attention in the field of data mining. Some researchers call it filter models, while other classify it as wrapped around methods. It is also known as subset selection in the statistics community.

Since the computational cost of brute-force feature selection method is prohibitively high, with considerable danger of overfitting, people resort to greedy methods, such as forward selection. We must first clarify the problem of performance evaluation of a set of input features. Even if the feature sets are evaluated by test-set cross-validation or leave-one-out cross validation, an exhaustive search of possible feature sets is likely to find a misleadingly well scoring feature set by chance. To prevent this from happening, we can use a cascaded cross-validation procedure, which selects from increasingly large sets of features (and thus from increasingly large model classes).

1. Shuffle the data set and split into a training set of 70% of the data and a test-set of the remaining 30%.
2. Let j vary among feature set sizes $j = (0, 1, .., m)$
   (a) Let f_{s_j} = best feature set of j, where best is measured as the minimiser of the leave-one-out cross-validation error over the training set.
   (b) Let $Testscore_j$ = the RMS prediction error of feature set f_{s_j} on the test-set.
3. Select the feature set f_{s_j} for which the test-set score is minimised.

Electronic copy available at: https://ssrn.com/abstract=3307566

The score for the best feature set of a given size is computed by independent cross-validation from the score for the best size of feature set. Note, this procedure does not describe how the search for the best feature set of size j in step 2a) is done. Further, the performance evaluation of a feature selection algorithm is more complicated than the evaluation of the feature set. This is because we must first ask the algorithm to find the best feature subset. We must then give a fair estimate of how well the feature selection algorithm performs by trying the first step on different data sets. Hence, the full procedure of evaluating the performance of a feature selection algorithm (described above) should have two layers of loops (see algorithm below). The inner loop using an algorithm to find the best subset of features, and the outer loop evaluating the performance of the algorithm with different data sets.

1. Collect a training data set from the specific domain.
2. Shuffle the data set.
3. Break it into P partition, say $P = 20$.
4. For each partition ($i = 0, 1, .., P - 1$)
   (a) Let $OuterTrainset(i)$ = all partitions except i.
   (b) Let $OuterTestset(i)$ = the ith partition.
   (c) Let $InnerTrain(i)$ = randomly chosen 70% of the $OuterTrainset(i)$.
   (d) Let $InnerTest(i)$ = the remaining 30% of the $OuterTrainset(i)$.
   (e) For $j = 0, 1, .., m$
   Search for the best feature set with j components $f_{s_{ij}}$ using leave-one-out on $InnerTrain(i)$.
   Let $InnerTestScore_{ij}$ = RMS score of $f_{s_{ij}}$ on $InnerTest(i)$
   End loop of (j).
   (f) Select the $f_{s_{ij}}$ with the best inner test score.
   (g) Let $OuterScore_i$ = RMS score of the selected feature set on $OuterTestset(i)$.

   End of loop of (i).
5. Return the Mean Outer Score

We are now going to introduce the forward feature selection algorithm, and explore three greedy variants of this algorithm improving the computational efficiency without sacrificing too much accuracy.

The forward feature selection procedure begins by evaluating all feature subsets consisting of only one input attribute. We start by measuring the leave-one-out cross-validation (LOOCV) error of the one-component subsets $\{X_1\}, \{X_2\}, ..\{X_M\}$, where M is the input dimensionality, so that we can find the best individual feature $X_{(1)}$. Next, the forward selection finds the best subset consisting of two components, $X_{(1)}$, and one other feature from the remaining $M - 1$ input attributes. Hence, there are a total of $M - 1$ pairs. We let $X_{(2)}$ be the other attribute in the best pair besides $X_{(1)}$. Then, the input subsets with three, four, and more features are evaluated. According to the forward selection, the best subset with m features is the m-tuple consisting of $\{X_1\}, \{X_2\}, ..\{X_m\}$, while overall, the best feature set is the winner out of all the M steps. Assuming that the cost of a LOOCV evaluation with i features is $C(i)$, then the computational cost of forward selection searching for a feature subset of size m out of M total input attributes will be

$$MC(1) + (M - 1)C(2) + ... + (M - m + 1)C(m)$$

For example, the cost of one prediction with one-nearest neighbour as the function approximator, using a kd-tree with j inputs, is $\mathcal{O}(j \log N)$ where N is the number of data points. Thus, the cost of computing the mean leave-one-out error,

Electronic copy available at: https://ssrn.com/abstract=3307566

which involves N predictions, is $\mathcal{O}(jN \log N)$. Hence, the total cost of feature selection using the above formula is $\mathcal{O}(m^2 MN \log N)$.

An alternative solution to finding the total best input feature set is to employ exhaustive search. This method starts with searching the best one-component subset of the input features, which is the same in the forward selection algorithm. Then, we need to find the best two-component feature subset which may consist of any pairs of the input features. Afterwards, it consists in finding the best triple out of all the combinations of any three input features, etc. One can see that the cost of exhaustive search is

$$MC(1) + \binom{M}{2} C(2) + ... + \binom{M}{m} C(m)$$

and we see that the forward selection is much cheaper than the exhaustive search. However, the forward selection may suffer due to its greediness. For example, if $X_{(1)}$ is the best individual feature, it does not guarantee that either $\{X_{(1)}, X_{(2)}\}$ or $\{X_{(1)}, X_{(3)}\}$ must be better than $\{X_{(2)}, X_{(3)}\}$. As a result, a forward selection algorithm may select a feature set different from that selected by the exhaustive searching. Hence, with a bad selection of the input features, the prediction Y_q of a query $X_q = \{x_1, x_2, ..., x_M\}$ may be significantly different from the true Y_q.

In the case where the greediness of forward selection does not have a significantly negative effect on accuracy, we need to know how to modify the forward selection algorithm to be greedier in order to further improve the efficiency. There exists several greedier feature selection algorithms whose goal is to select no more than m features from a total of M input attributes, and with tolerable loss of prediction accuracy. We briefly discuss three of these algorithms

1. The super greedy algorithm: do all the 1-attribute LOOCV calculations, sort the individual features according to their LOOCV mean error, then take the m best features as the selected subset. Thus, we do M computations involving one feature and one computation involving m features. If the nearest neighbour is the function approximator, the cost of super greedy algorithm is $\mathcal{O}((M + m)N \log N)$.

2. The greedy algorithm: do all the 1-attribute LOOCV and sort them, take the best two individual features and evaluate their LOOCV error, then take the best three individual features, and so on, until m features have been evaluated. Compared with the super greedy algorithm, this algorithm may conclude at a subset whose size is smaller than m but whose inner tested error is smaller than that of the m component feature set. Hence, the greedy algorithm may end up with a better feature set than the super greedy one does. The cost of the greedy algorithm for the nearest neighbour is $\mathcal{O}((M + m^2)N \log N)$.

3. The restricted forward selection (RFS):

   (a) caculate all the 1-feature set LOOCV errors, and sort the features according to the corresponding LOOCV errors. We let the features ranking from the most important to the least important be $X_{(1)}, X_{(2)}, ..., X_{(M)}$.

   (b) do the LOOCV of 2-feature subsets consisting of the winner of the first round, $X_{(1)}$, along with another feature, either $X_{(2)}$ or $X_{(3)}$, or any other one until $X_{(\frac{M}{2})}$. There are $\frac{M}{2}$ of these pairs. The winner of this round will be the best 2-component feature subset chosen by RFS.

   (c) calculate the LOOCV errors of $\frac{M}{3}$ subsets consisting of the winner of the second round, along with the other $\frac{M}{3}$ features at the top of the remaining rank. In this way, the RFS will select its best feature triple.

   (d) continue this procedure until RFS has found the best m-component feature set.

   (e) From step 1) to step 4), the RFS has found m feature sets whose sizes range from 1 to m. By comparing their LOOCV errors, the RFS can find the best overall feature set.

One of the difference between RFS and conventional forward selection (FS) is that at each step, when inserting an additional feature into the subset, the FS considers all the remaining features, while the RFS only tries the part of them which seems the more promising. The cost of RFS for the nearest neighbour is $\mathcal{O}(MmN \log N)$.

Electronic copy available at: https://ssrn.com/abstract=3307566

We are left with finding out how cheap and how accurate all these varieties of forward selection are compared with the conventional forward selection method. Using real world data sets from StatLib/CMU and UCI's machine learning data repository coming from different domains such as biology, sociology, robotics etc, it was demonstrated that Exhaustive Search (ES) is prohibitively time consuming. Even though the features selected by FS may differ from the result of ES because some of the input features are not mutually independent, ES is far more expensive than the FS, while it is not significantly more accurate than FS. In order to investigate the influence of greediness on the above three greedy algorithms, we consider

1. the probabilities for these algorithms to select any useless features
2. the prediction errors using the feature set selected by these algorithms
3. the time cost for these algorithms to find their feature sets

It was found that FS does not eliminate more useless features than the greedier competitors except for the Super Greedy one. However, the greedier the algorithm, the more easily confused by the relevant but corrupted features it becomes. Further, the three greedier feature selection algorithms do not suffer great loss in accuracy, and RFS performs almost as well as the FS. As expected, the greedier algorithms improve the efficiency. It was found that the super greedy algorithm (Super) was ten time faster than the FS, while the greedy algorithm (Greedy) was seven times faster, and the restricted forward selection (RFS) was three times faster. Finally, the RFS performed better than the conventional FS in all aspects. Inserting more independent random noise and corrupted features to the data sets, the probability for any corrupted feature to be selected remained almost the same, while that of independent noise reduced greatly. To conclude, while in theory the greediness of feature selection algorithms may lead to great reduction in the accuracy of function approximation, in practice it does not often happen. The three greedier algorithms discussed above improve the efficiency of the forward selection algorithm, especially for larger data sets with high input dimensionalities, without significant loss in accuracy. Even in the case where the accuracy is more crucial than the efficiency, restricted forward selection is more competitive than the conventional forward selection.

2.2 Introduction to data mining

2.2.1 From data to information

We refer the readers to textbooks by Hand et al. [2001], Han et al. [2006], among others.

2.2.1.1 Data

2.2.1.1.1 Definitions Data is a set of values of qualitative or quantitative variables. In computing, data is information that has been translated into a form that is efficient for movement or processing. A digital computer represents a piece of data as a sequence of symbols drawn from a fixed alphabet. The most common digital computers use a binary alphabet, that is, an alphabet of two characters, typically denoted 0 and 1. Thus, data is information converted into binary digital form. It is acceptable for data to be used as a singular subject or a plural subject. More generally, data, information, knowledge and wisdom are closely related concepts, but each has its own role in relation to the other, and each term has its own meaning. Data is collected and analysed; data only becomes information suitable for making decisions once it has been analysed in some fashion. The concept of data in the context of computing was introduced by Claude Shannon who ushered in binary digital concepts based on applying two-value Boolean logic to electronic circuits (see Section (2.1)). The amount of information content in a data stream may be characterised by its Shannon entropy. Data is often assumed to be the least abstract concept, information the next least, and knowledge the most abstract. Thus, data becomes information by interpretation (see Ilkka [2000]). Generally, the concept of information is closely related to notions of constraint, communication, control, data, form, instruction, knowledge, meaning, mental stimulus, pattern, perception, and representation (see Beynon-Davies [2009]). This leads to the concept of data mining.

Electronic copy available at: https://ssrn.com/abstract=3307566

2.2.1.1.2 Data set A data set consists of feature vectors, where each feature vector is a description of an object by using a set of features. For example, in a three-Gaussian data set, a feature vector can be represented by $(value1, value2, cross)$ or $(value1, value2, circle)$, where $(value1, value2)$ are (x, y)-coordinates and $cross$ and $circle$ are labels. The number of features of a data set is called dimension. Features are also called attributes, a feature vector is also called an instance, and a data is sometime called a sample. Labelled data is a group of samples that have been tagged with one or more labels. Labelling typically takes a set of unlabelled data and augments each piece of that unlabelled data with meaningful tags that are informative. A data class refers to a class that contains only fields and crude methods for accessing them (getters and setters). These are simply containers for data used by other classes.

2.2.1.2 Data mining

Data mining is the process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems. It is an interdisciplinary subfield of computer science with an overall goal of extracting information (with intelligent method) from a data set and transform the information into a comprehensible structure for further use (see Hastie et al. [2009], Kamber et al. [2011]). The rapid growth and integration of databases provided scientists, engineers, and business people with a vast new resource that can be analysed to make scientific discoveries, optimise industrial systems, and uncover financially valuable patterns. New methods targeted at large data mining problems have been developed. Hand et al. [2001]) defined Data Mining (DM) as follows:

Definition 2.2.1 *Data Mining is the analysis of (large) observational data sets to find unsuspected relationships and to summarise the data in novel ways that are both understandable and useful to the data owner.*

More generally, data mining is the analysis step of the Knowledge Discovery in Databases process, or KDD (see Han et al. [2006]). The goal is the extraction of patterns and knowledge from large amounts of data, not the extraction (mining) of data itself. According to the Cross Industry Standard Process for Data Mining (CRISP-DM), the six phases of the knowledge discovery in databases (KDD) process are

- Business understanding
- Data understanding
- Data preparation
- Modelling
- Evaluation
- Deployment

or a simplified process such as (1) Pre-processing, (2) Data Mining, and (3) Results Validation.
Before data mining algorithms can be used, a target data set must be assembled. A common source for data is a data mart or data warehouse. Pre-processing is essential to analyse the multivariate data sets before data mining. The target set is then cleaned. Data cleaning removes the observations containing noise and those with missing data.
There exists several functionalities to data mining, such as:

1. Data characterisation summarising the general characteristics or features of a target class of data.
2. Data discrimination comparing the general features of target class data objects with the general features of objects from a set of contrasting classes.
3. Association analysis is the discovery of association rules showing attribute value conditions occurring frequently together in a given set of data.

Electronic copy available at: https://ssrn.com/abstract=3307566

4. Classification is the process of finding a set of models (or functions) that describe and distinguish data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown.

In fact, data mining involves six common classes of tasks (see Sondwale [2015]):

1. Anomaly detection (outlier/change/deviation detection): The identification of unusual data records, that might be interesting or data errors that require further investigation.
2. Association rule learning (dependency modelling): Searches for relationships between variables. For example, a supermarket might gather data on customer purchasing habits. Using association rule learning, the supermarket can determine which products are frequently bought together and use this information for marketing purposes. This is sometimes referred to as market basket analysis.
3. Clustering: is the task of discovering groups and structures in the data that are in some way or another similar, without using known structures in the data.
4. Classification: is the task of generalising known structure to apply to new data. For example, an e-mail program might attempt to classify an e-mail as legitimate or as spam.
5. Regression: attempts to find a function which models the data with the least error that is, for estimating the relationships among data or datasets.
6. Summarisation: providing a more compact representation of the data set, including visualisation and report generation.

2.2.1.3 Modelling

In general, a model is a predictive model that we want to construct or discover from the data set. One major task of pattern recognition and data mining, within machine learning, is to construct good models from data sets (see Otero et al. [2013]). The process of generating models from data is called learning or training. There are several types of learning, such as

- supervised learning: it consists of a specified set of classes, and example objects labelled with the appropriate class. The goal being to learn from the training objects, enabling novel objects to be identified as belonging to one of the classes.
- unsupervised learning: it is the task of inferring a function that describes the structure of unlabelled data (data that has not been classified or categorised). Among neural network models, the self-organizing map (SOM) and adaptive resonance theory (ART) are commonly used in unsupervised learning algorithms.

The learned model is called a predictor or a hypothesis (or learner). There are several types of label such as categorical or numerical.

1. If the label is categorical, such as shape, the task is also called classification and the learner is also called classifier.
2. However, if the label is numerical, such as x-coordinate, the task is also called regression and the learner is also called fitted regression model.

If the label is categorical, such as shape, the task is called classification and the learner is called a classifier. If the label is numerical, such as x-coordinate, the task is called regression and the learner is called fitted regression model.
In this appendix, we will focus on supervised learning, especially classification.

Electronic copy available at: https://ssrn.com/abstract=3307566

2.2.2 Classification

2.2.2.1 Terminology

In the more general problem of pattern recognition, we distinguish

- Classification and clustering, which is the assignment of some sort of output value to a given input value
- Regression, which assigns a real-valued output to each input
- Sequence labelling, which assigns a class to each member of a sequence of values (for example, part of speech tagging, which assigns a part of speech to each word in an input sentence)
- Parsing, which assigns a parse tree to an input sentence, describing the syntactic structure of the sentence

Often, the individual observations are analysed into a set of quantifiable properties, known variously as explanatory variables or features. These properties may variously be

- categorical (such as A, B, AB or O, for blood type),
- ordinal (such as large, medium or small),
- integer-valued (such as the number of occurrences of a particular word in an email), or
- real-valued (such as a measurement of blood pressure).

Terminology across fields is quite varied. In statistics, where classification is often done with logistic regression or a similar procedure, the properties of observations are termed explanatory variables (or independent variables, regressors, etc.), and the categories to be predicted are known as outcomes, which are considered to be possible values of the dependent variable. In machine learning, the observations are often known as instances, the explanatory variables are termed features (grouped into a feature vector), and the possible categories to be predicted are classes.
Class label is the discrete attribute having finite values (dependent variable) whose value you want to predict based on the values of other attributes (features). The class label always takes on a finite number of different values. Classification is a type of problem whereas labelling is a function trying to label an object and classify using the information. For instance, given a set of examples of the form $(\text{attribute values}\ ,\ \text{class label})$, we want to learn a rule that computes the label from the attribute values. For example, in binary classification, we use positive $(+1)$ and negative (-1) to denote the two class labels.

2.2.2.2 Definitions

In machine learning and statistics, classification is the problem of identifying to which of a set of categories (subpopulations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known (see Alpaydin [2010]). Classification is considered an instance of supervised learning, that is, learning where a training set of correctly identified observations is available. The corresponding unsupervised procedure is known as clustering, and involves grouping data into categories based on some measure of inherent similarity or distance.
Machine learning have been traditionally used for classification. A classification process usually consists of three main stages:

1. In the first stage, data from objects have to be collected for the design of classifiers.
2. In the second stage, feature extraction is performed to extract characteristics from the collected data to be classified such that redundant information is removed and representative information is extracted resulting in reduction of input dimensions and improved classification accuracy.

Electronic copy available at: https://ssrn.com/abstract=3307566

3. In the third stage, a classifier is designed using the feature data.

Classification can be thought of as two separate problems:

1. binary classification: only two classes are involved (it is a better understood task).
2. multiclass classification: involves assigning an object to one of several classes. Since many classification methods have been developed specifically for binary classification, multiclass classification often requires the combined use of multiple binary classifiers.

2.2.2.3 The algorithms

Most algorithms describe an individual instance whose category is to be predicted using a feature vector of individual, measurable properties of the instance. Some algorithms work only in terms of discrete data and require that real-valued or integer-valued data be discretised into groups (e.g. less than 5, between 5 and 10, or greater than 10). Other classifiers work by comparing observations to previous observations by means of a similarity or distance function.
In the literature, classification techniques and methods from traditional methods to machine learning methods can be found. Examples of the former are Linear discriminant analysis (LDA), logic based method, statistical approach, or instance-based methods. In the latter, models such as decision trees, SVM, neural networks, Bayesian belief networks, genetic algorithm etc have been considered. The ability to perform classification and be able to learn to classify objects is paramount to the process of decision making.
The process of seeking relationships within a data set involves a number of steps:

1. Model or pattern structure: determining the nature and structure of the representation to be used.
   (a) a model structure is a global summary of a data set making statements about any point in the full measurement space,
   (b) pattern structures only make statements about restricted regions of the space spanned by the variables.
2. Score function: deciding how to quantify and compare how well different representations fit the data (choosing a score function).
   Score functions judge the quality of a fitted model, and should precisely reflect the utility of a particular predictive model.
3. Optimisation and search method: choosing an algorithmic process optimising the score function.
   (a) optimisation problem: the task of finding the best values of parameters in models
   (b) combinatorial problem: the task of finding interesting patterns (such as rules) from a large family of potential patterns, and is often accomplished using heuristic search techniques.
4. Data management strategy: deciding what principles of data management are required for implementing the algorithms efficiently.
   Data management strategy is about the ways in which the data are stored, indexed, and accessed.

Some examples of classification algorithms include:

- Linear classifiers: Fisher's linear discriminant (see Fisher [1936]), Logistic regression, Naive Bayes classifier (see Binder [1978]), Perceptron
- Support vector machines: Least squares support vector machines
- Quadratic classifiers

Electronic copy available at: https://ssrn.com/abstract=3307566

- Kernel estimation: k-nearest neighbour
- Boosting (meta-algorithm)
- Decision trees: Random forests
- Neural networks
- Learning vector quantization

The success of classification learning is heavily dependent on the quality of the data provided for training, as the learner only has the input to learn from. On the other hand, we want to avoid overfitting the given data set, and would rather find models or patterns generalising potential future data. Note, even though data mining is an interdisciplinary exercise, it is a process relying heavily on statistical models and methodologies. The main difference being the large size of the data sets to manipulate, requiring sophisticated search and examination methods. Further difficulties arise when there are many variables (curse of dimensionality), and often the data is constantly evolving. Recently, a lot of advances have been made on machine learning strategies mimicking human learning, and we refer the reader to Battula et al. [2013] for more details.

2.2.3 The challenges of computational learning

While the ability to perform classification and be able to learn to classify objects is paramount to the process of decision making, there exists several types of learning (see Mitchell [1997]), such as

- Supervised learning: learning a function from example data made of pairs of input and correct output.
- Unsupervised learning: learning from patterns without corresponding output values
- Reinforcement learning: learning with no knowledge of an exact output for a given input. Nonetheless, online or delayed feedback on the desirability of the types of behaviour can be used to help adaptation of the learning process.
- Active learning: learning through queries and responses.

More formally, these types of learning are part of what is called inductive learning where conclusions are made from specific instances to more general statements. That is, examples are provided in the form of input-output pairs $[X, f(X)]$ and the learning process consists of finding a function h (called hypothesis) which approximates a set of samples generated by the function f. The search for the function h is formulated in such a way that it can be solved by using search and optimisation algorithms.
Some of the challenges of computational learning can be summarised as follow:

- Identifying a suitable hypothesis can be computationally difficult.
- Since the function f is unknown, it is not easy to tell if the hypothesis h generated by the learning algorithm is a good approximation.
- The choice of a hypothesis space describing the set of hypotheses under consideration is not trivial.

As a result, a simple hypothesis consistent with all observations is more likely to be correct than a complex one. In the case where multiple hypotheses (an Ensemble) are generated, it is possible to combine their predictions with the aim of reducing generalisation error. For instance, boosting works as follow

- Examples in the training set are associated with different weights.

Electronic copy available at: https://ssrn.com/abstract=3307566

- The weights of incorrectly classified examples are increased, and the learning algorithm generates a new hypothesis from this new weighted training set. The process is repeated with an associated stopping criterion.
- The final hypothesis is a weighted-majority of all the generated hypotheses which can be based on different mixture of expert rules.

The difficult part being to know when to stop the iterative process and how to define a proper measure of error. One way forward is to consider the Probably Approximately Correct (PAC) learning which can be described as follow:

- A hypothesis is called approximately correct if its error insample lies within a small constant of the true error.
- By learning from a sufficient number of examples, one can calculate if a hypothesis has a high probability of being approximately correct.
- There is a connection between the past (seen) and the future (unseen) via an assumption stating that the training and test datasets come from the same probability distribution. It follows from the common sense that non-representative samples do not help learning.

More formally, a concept class C is said to be PAC learnable using a hypothesis class H if there exists a learning algorithm L such that for all concepts in C, for all instance distributions D on an instance space X,

$$\forall \epsilon \, , \, \delta \text{ such that } 0 < \epsilon \, , \, \delta < 1$$

when given access to the example set, produces with probability at least $(1 - \delta)$, a hypothesis h from H with error no-more than ϵ. To specify the problem we get a set of instances X, a set of hypotheses H, a set of possible target concepts C, training instances generated by a fixed, unknown probability distribution $\mathcal{D}$ over X, a target value $c(x)$, some training examples $< x, c(x) >$, and a hypothesis h estimating c. Then, the error of a hypothesis h satisfies

$$error_D = P_{x \in \mathcal{D}}(c(x) \neq h(x))$$

and the deviation of the true error from the training error satisfies

- Training error: $h(x) \neq c(x)$ over training instances.
- True error: $h(x) \neq c(x)$ over future random instances.

We must now measure the difference between the true error and the training error. Any hypothesis h is consistent when for all training samples,

$$h(x) = c(x)$$

If the hypothesis space H is finite, and $\mathcal{D}$ is a sequence of $m \geqslant 1$ independent random examples of some target concept c, then for any $0 \leqslant \epsilon \leqslant 1$, the probability that $VS_{H,D}$ contains a hypothesis with error greater than ϵ is less than

$$|H|e^{-\epsilon m}$$

and $P(1 \text{ of } |H| \text{ hyps. consistent with m exs.}) < |H|e^{-\epsilon m}$. Considering a bounded sample size, for the probability to be at most δ, that is, $|H|e^{-\epsilon m} \leqslant \delta$, then

$$m \geqslant \frac{1}{\epsilon}\big(\ln |H| + \ln \frac{1}{\delta}\big)$$

More can be found on agnostic learning and infinite hypothesis space in books by Mitchell [1997] and Bishop [2006].

Electronic copy available at: https://ssrn.com/abstract=3307566

2.2.4 Evaluation and validation

Data mining can unintentionally be misused, and can then produce results which appear to be significant, but which do not actually predict future behaviour and cannot be reproduced on a new sample of data and bear little use. Often this results from investigating too many hypotheses and not performing proper statistical hypothesis testing. Not all patterns found by the data mining algorithms are necessarily valid. It is common for the data mining algorithms to find patterns in the training set which are not present in the general data set. This is called overfitting. To overcome this, the evaluation uses a test set of data on which the data mining algorithm was not trained. The learned patterns are applied to this test set, and the resulting output is compared to the desired output. A number of statistical methods may be used to evaluate the algorithm, such as Precision and Recall which are evaluated from True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). More recently Sensitivity, Specificity, Accuracy and Receiver Operating Characteristic (ROC) curves have been used to evaluate the tradeoff between true- and false-positive rates of classification algorithms.
Sensitivity and Specificity are statistical measures of the performance of a binary classification test, also known in statistics as a classification function:

- Sensitivity (also called the true positive rate, the recall, or probability of detection in some fields) measures the proportion of actual positives that are correctly identified as such (e.g., the percentage of sick people who are correctly identified as having the condition).
- Specificity (also called the true negative rate) measures the proportion of actual negatives that are correctly identified as such (e.g., the percentage of healthy people who are correctly identified as not having the condition).

In the terminology true/false positive/negative, true or false refers to the assigned classification being correct or incorrect, while positive or negative refers to assignment to the positive or the negative category.
In medicine sensitivity, specificity and accuracy are widely used statistics to describe a diagnostic test since they quantify how good and reliable a test is.

- Sensitivity evaluates how good the test is at detecting a positive disease
- Specificity estimates how likely patients without disease can be correctly ruled out
- The ROC curve is a graphic presentation of the relationship between both sensitivity and specificity helping to decide the optimal model through determining the best threshold for the diagnostic test
- Accuracy measures how correct a diagnostic test identifies and excludes a given condition. Accuracy of a diagnostic test can be determined from sensitivity and specificity with the presence of prevalence.

The terms used are true positive (TP), true negative (TN), false negative (FN), and false positive (FP). If a disease is proven present in a patient, the given diagnostic test also indicates the presence of disease, the result of the diagnostic test is considered true positive. Similarly, if a disease is proven absent in a patient, the diagnostic test suggests the disease is absent as well, the test result is true negative (TN). See details in Table (2.1). Both true positive and true negative suggest a consistent result between the diagnostic test and the proven condition (also called standard of truth). If the diagnostic test indicates the presence of disease in a patient who actually has no such disease, the test result is false positive (FP). Similarly, if the result of the diagnosis test suggests that the disease is absent for a patient with disease for sure, the test result is false negative (FN). Both false positive and false negative indicate that the test results are opposite to the actual condition.

Electronic copy available at: https://ssrn.com/abstract=3307566

Outcome of the test	**Condition as determined by the Standard of Truth**		
	Positive	Negative	Row total
Positive	TP	FP	$TP + FP$
Negative	FN	TN	$FN + TN$
Column total	$TP + FN$	$FP + TN$	$N = TP + TN + FP + FN$

Table 2.1: Accuracy table

The evaluation measures are described in terms of TP, TN, FN and FP. The Precision and Recall are defined as follows:

$$\begin{aligned} \text{Precision}_{+} &= \frac{TP}{TP + FP} \\ \text{Precision}_{-} &= \frac{TN}{TN + FN} \\ \text{Recall}_{+} &= \frac{TP}{TP + FN} \\ \text{Recall}_{-} &= \frac{TN}{TN + FP} \end{aligned}$$

Precision is the weighted average of precision positive and negative, while Recall is the weighted average of recall positive and negative. The F-measure is given by

$$\text{F-measure} = \frac{2 \times \text{ Precision } \times \text{ Recall}}{\text{Precision } + \text{ Recall}}$$

The value range of these measures is between 0 and 1, where 0 indicates the worst performance, while 1 indicates the best performance.
Sensitivity, Specificity and Accuracy are defined as follows:

$$\begin{aligned} \text{Sensitivity} &= \frac{TP}{TP + FN} \\ \text{Specificity} &= \frac{TN}{TN + FP} \\ \text{Accuracy} &= \frac{TN + TP}{TN + TP + FN + FP} \end{aligned}$$

Thus, sensitivity is the proportion of true positives that are correctly identified by a diagnostic test, showing how good the test is at detecting a disease. The numerical values of sensitivity represents the probability of a diagnostic test identifies patients who do in fact have the disease. The higher the numerical value of sensitivity, the less likely diagnostic test returns false-positive results. Specificity is the proportion of the true negatives correctly identified by a diagnostic test, suggesting how good the test is at identifying normal (negative) condition. The numerical value of specificity represents the probability of a test diagnoses a particular disease without giving false-positive results. Accuracy is the proportion of true results, either true positive or true negative, in a population. It measures the degree of veracity of a diagnostic test on a condition.
Note, accuracy can be determined from sensitivity and specificity, where prevalence is known. Prevalence is the probability of disease in the population at a given time

$$\text{Accuracy } = \text{ Sensitivity } \times \text{ Prevalence } + \text{ Specificity } \times \big(1 - \text{ Prevalence }\big)$$

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, even if both sensitivity and specificity are high, say 99%, it does not suggest that the accuracy of the test is equally high as well.
For a given diagnostic test, the true positive rate (TPR) against false positive rate (FPR) can be measured, where $TRP = \frac{TP}{TP+FN}$ and $FPR = \frac{FP}{FP+TN}$. Thus, TPR is equivalent to sensitivity and FPR is equivalent to specificity. All possible combinations of TPR and FPR compose a ROC space. One TPR and one FPR together determine a single point in the ROC space, and the position of a point in the ROC space shows the tradeoff between sensitivity and specificity, that is, the increase in sensitivity is accompanied by a decrease in specificity. Thus the location of the point in the ROC space depicts whether the diagnostic classification is good or not. In an ideal situation, a point determined by both TPR and FPF yields a coordinates $(0, 1)$, or we can say that this point falls on the upper left corner of the ROC space. This idea point indicates the diagnostic test has a sensitivity of 100% and specificity of 100%. It is also called perfect classification. Diagnostic test with 50% sensitivity and 50% specificity can be visualised on the diagonal determined by coordinate $(0, 0)$ and coordinates $(1, 0)$. Theoretically, a random guess would give a point along this diagonal. A point predicted by a diagnostic test fall into the area above the diagonal represents a good diagnostic classification, otherwise a bad prediction.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 3

Ensemble models

We briefly introduce ensemble models. More details can be found in textbooks by Rokach [2010], Zhou [2012], among others.

3.1 The base models

We let $\mathcal{X}$ and $\mathcal{Y}$ be the input and output spaces, $\mathcal{D}$ the probability distribution, D the data set (sample), H the set of hypotheses, $I(\cdot)$ the indicator function. We let x be an instance or feature vector. For example, we consider the training data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$ where the instances x_i are independent and identically distributed (i.i.d.) and $y_i = f(x_i)$ where f is the ground-truth target function. That is, x_i is a vector and y_i takes value in $\mathbb{R}$. The learning process aim at constructing a learner h minimising the generalisation error

$$\mathcal{E}(h) = E_x[I(h(x) \neq f(x))]$$

We can also let $\mathcal{X}$ be the instance space, $\mathcal{Y}$ be the label space, define the labelled data set as $L = \{(x_1, y_1), ..., (x_l, y_l)\}$ and consider the binary classification tasks where $\mathcal{Y} = \{-1, +1\}$.

3.1.1 Introduction

3.1.1.1 Presentation

Ensemble methods train multiple learners to solve the same problem by combining them. It contains a number of learners called base learners generated from a base learning algorithm (decision tree, neural network etc). The base learners are also referred to as weak learners. A weak learner is just slightly better than random guess, while a strong learner is very close to perfect performance.
Early contributions to ensemble methods are:

- combining classifiers (for pattern recognition working on strong classifiers),
- ensembles of weak learners (machine learning boosting performance from weak to strong, such as, AdaBoost, Bagging, etc), and
- mixture of experts (neural networks considering a divide-and-conquer strategy).

Hansen et al. [1990] found that predictions made by the combination of a set of classifiers are often more accurate than the ones made by the best single classifier. Further, Schapire [1990] proved that weak learners (easy to obtain)

Electronic copy available at: https://ssrn.com/abstract=3307566

can be boosted to strong learners.
Stacking (also called stacked generalisation) proposed by Wolpert [1992] is a popular method for improving results on data mining competitions. It is a model ensembling technique used to combine information from multiple predictive models to generate a new model (see Figure (3.1)). It is assumed that the stacked model (also called 2nd-level model) will outperform each of the individual models due to its smoothing nature and ability to highlight each base model where it performs best and discredit the ones where it poorly performs. Thus, stacking is most effective when the base models are significantly different. Some of the advantages of the stacking techniques are to avoid over-fitting due to K fold cross validation and to interpret non-linearity between features due to treating output scores as features.

3.1.1.2 Definitions

We briefly introduce ensemble methods and then present a few facts on model stacking.

Definition 3.1.1 *An ensemble is a set of classifiers that learn a target function, and their individual predictions are combined to classify new examples.*

An ensemble is constructed in two steps, generating the base learners, and then combining them. There are two basic ways of aggregating classification methods:

1. Before the fact: Creates solutions to be combined (for example bagging).
   Suppose we want to create a robust method for determining if a given image contains a face:
   - Problem:It may be very difficult and computationally intensive to create a classifier for that task
   - One solution: detect eyes instead of faces
     - Advantage: A lot more efficient
     - Problem: this may produce models with low accuracy
     - Solution: Combine it with other similar classifiers
2. After the fact: Combines existing solutions (for example blending).
   Netflix challenge teams merging: Different models are already built. Find an intelligent way of combining them. Given each hypothesis $\widehat{f}_1, ..., \widehat{f}_M$ and a new instance x, we can use a linear regression to compute the prediction $g(x)$. That is,

$$\widehat{f}_1, ..., \widehat{f}_M \rightarrow g(x) = \sum_{i=1}^{M} \omega_i \widehat{f}_i(x)$$

   Choose ω_i to minimise the error on aggregation set.

Each of these methods can be more or less appropriate to particular problems. There are two paradigms of ensemble methods

1. sequential ensemble methods: the base learners are generated sequentially (AdaBoost). It exploit the dependence between the base learners, since the overall performance can be boosted in a residual-decreasing way.
2. parallel ensemble methods: the base learners are generated in parallel (Bagging). It exploit the independence between the base learners, since the error can be reduced dramatically by combining independent base learners.

Some examples of ensemble methods are bagging, boosting, random forests, extra trees. Nowadays, stacking and blending are also ensemble methods (see details in textbook by Zhou [2012]). We now briefly introduce stacking:

Definition 3.1.2 *Model stacking is a learning method that aims to improve predictive accuracy by combining predictions from multiple models. It is a particular case of ensemble learning.*

Electronic copy available at: https://ssrn.com/abstract=3307566

It is argued that model stacking leads to better predictions, at the expense of interpretability [1]. In general, it is used as a component of several winning entries in Kaggle competitions.

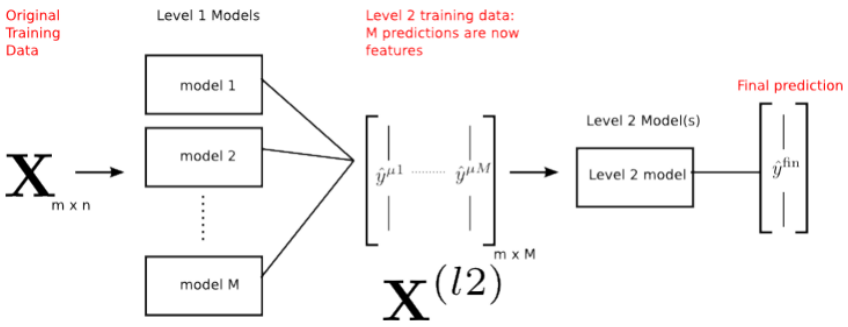


Figure 3.1: Two levels model stacking

3.1.1.3 Diversity

We call ensemble diversity the difference among the individual learners. The diversity is crucial to ensemble performance since a gain from combination requires the individual learners to be different. The major problem is that the individual learners are trained for the same task from the same training data, and thus they are usually highly correlated. However, several algorithms assume either independent or less correlated learners. Even if correlation between individual learners is considered, it is generally based on the assumption that the individual learners produce independent estimates of the posterior probabilities, which is not the case in practice. In addition, the problem of generating diverse individual learners is even more challenging if we consider that the individual learners must not be very poor, and otherwise their combination would not improve and could even worsen the performance. For example, when the performance of individual classifiers is quite poor, the added error of simple soft voted ensemble may become arbitrarily large. As a result, it is desired that the individual learners should be accurate and diverse. Nonetheless, combining only accurate learners is often worse than combining some accurate ones together with some relatively weak ones, since complementarity is more important than pure accuracy. In conclusion, the success of ensemble learning lies in achieving a good tradeoff between the individual performance and diversity. Unfortunately, there is no well-accepted formal definition of diversity, but it is crucial in ensemble learning.

3.1.1.4 Generalisation error

Ensemble methods are based on statistical tools and, as such, rely on error computation and error reduction. For instance, we take binary classification on classes $\{-1, +1\}$ and let f be the truth function. We suppose that each base classifier has an independent generalisation error ϵ, that is, for base classifier h_i

$$P(h_i(x) \neq f(x)) = \epsilon$$

After combining N number of such base classifiers according to

$$H(x) = \text{sgn}\big(\sum_{i=1}^{N} h_i(x)\big)$$

the ensemble H makes an error only when at least half of its base classifiers make errors. By Hoeffding inequality, the generalisation error of the ensemble is

[1] In mathematical logic, interpretability is a relation between formal theories that expresses the possibility of interpreting or translating one into the other.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$P(H(x) \neq f(x)) = \sum_{k=0}^{\lfloor \frac{N}{2} \rfloor} \binom{N}{k} (1-\epsilon)^k \epsilon^{N-k} \leqslant e^{-\frac{1}{2}N(2\epsilon-1)^2}$$

It shows that the generalisation error reduces exponentially to the ensemble size N and ultimately approaches to zero as N approaches to infinity. Even though one can not get absolutely independent base learners generated from the same training data set, base learners with less dependence can be obtained by introducing randomness in the learning process, and a good generalisation ability can be expected by the ensemble. Further, parallel methods are favourable to parallel computing, speeding up the training process.
One can show that the generalisation error of an ensemble depends on a term related to diversity. Thus, one must estimate error decomposition to understand ensemble learning. Examples are the error-ambiguity decomposition and the bias-variance decomposition.

3.1.2 Some simple models

3.1.2.1 Decision tree

Decision tree (DT) learning is one of the most popular techniques for classification. The classification model learnt through these techniques is represented as a tree called a decision tree. It consists in a set of tree-structured decision tests working in a divide-and-conquer way as follows:

- Each non-leaf node is associated with a feature test also called a split, which splits data into different subsets according to their different values on the feature test.
- Each leaf node is associated with a label, which will be assigned to instances falling into this node.

When predicting, a series of feature tests is conducted starting from the root node, and the result is obtained when a leaf node is reached. On the other hand, the classification process starts by testing whether the value of the feature y-coordinate is larger than a threshold; if so, the instance is classified as "cross", and otherwise the tree tests whether the feature value of x-coordinate is larger than another threshold; if so, the instance is classified as "cross" and otherwise is classified as "circle". The process is recursive since in each step, a data set is given and a split is selected, then this split is used to divide the data set into subsets, and each subset is considered as the given data set for the next step. Remains to decide how to select the splits. For instance, dividing the training set D into subsets $D_1, ..., D_k$, Breiman et al. [1984] proposed the CART which uses Gini index for selecting the split maximising the Gini

$$G_{gini}(D; D_1, ..., D_k) = I(D) - \sum_{i=1}^{k} \frac{|D_k|}{|D|} I(D_k)$$

where

$$I(D) = 1 - \sum_{y \in \mathcal{Y}} P^2(y|D)$$

Alternatively, Quinlan [1998] proposed to use the information gain criterion in the ID3 algorithm. Given a training set D, the entropy of D is defined as

$$\mathrm{Ent}(D) = -\sum_{y \in \mathcal{Y}} P(y|D) \log P(y|D)$$

If the training set D is divided into subsets $D_1, ..., D_k$, the entropy may be reduced. The amount of the reduction is the information gain

Electronic copy available at: https://ssrn.com/abstract=3307566

$$G(D; D_1, ..., D_k) = \text{Ent}(D) - \sum_{i=1}^{k} \frac{|D_k|}{|D|} \text{Ent}(D_k)$$

Thus, the feature-value pair which will cause the largest information gain is selected for the split. However, features with a lot of possible values will be favoured, disregarding their relevance to classification. Note, Quinlan [1993] had considered a variant of the information gain called the $C4.5$ based on the gain ratio

$$P(D; D_1, ..., D_k) = G(D; D_1, ..., D_k) \cdot \Big(\sum_{i=1}^{k} \frac{|D_k|}{|D|} \log \frac{|D_k|}{|D|} \Big)^{-1}$$

which takes normalisation on the number of feature values. In that setting, the feature with the highest gain ratio, among features with better-than-average information gains, is selected as the split. Note, $C4.5$ and CART can deal with numerical features.
To reduce the risk of overfitting, a general strategy is to employ pruning to cut off some tree branches caused by noise or peculiarities of the training set. Pre-pruning tries to prune branches when the tree is being grown, while post-pruning re-examines fully grown trees to decide which branches should be removed. This is achieved with the validation error: for pre-pruning, a branch will not be grown if the validation error will increase by growing the branch; for post-pruning, a branch will be removed if the removal will decrease the validation error.
Trees that are grown very deep tend to learn highly irregular patterns, but they overfit their training sets. That is, they have low bias, but very high variance.

3.1.2.2 Boosting

Boosting is an algorithm that converts weak learners to strong learners. We suppose the weak learner will work on any data distribution it is given, and take the binary classification task as an example. That is, we want to classify instances as positive and negative. The training instances in space $\mathcal{X}$ are drawn i.i.d. from distribution $\mathcal{D}$, and the truth function is f. Suppose the space X is made of three parts $\mathcal{X}_i$, $i = 1, 2, 3$, each takes $\frac{1}{3}$ amount of the distribution, and a learner working by random guess has 50% classification error on this problem. We want to get an accurate classifier on the problem, but can only have a weak classifier with correct classifications in spaces $\mathcal{X}_1$ and $\mathcal{X}_2$ and has wrong classifications in $\mathcal{X}_3$, thus has $\frac{1}{3}$ classification error. We denote this weak classifier as h_1. The idea of boosting is to correct the mistakes made by h_1 by deriving a new distribution $\mathcal{D}^{'}$ from $\mathcal{D}$, which makes the mistakes of h_1 more evident. For example, it focuses more on the instances in X_3. Then, we can train a classifier h_2 from $\mathcal{D}^{'}$. Again, suppose h_2 is also a weak classifier, which has correct classifications in $\mathcal{X}_1$ and $\mathcal{X}_3$ and has wrong classifications in $\mathcal{X}_2$. By combining h_1 and h_2 in an appropriate way, the combined classifier will have correct classifications in $\mathcal{X}_1$, and maybe some errors in $\mathcal{X}_2$ and $\mathcal{X}_3$. Again, we derive a new distribution $\mathcal{D}^{''}$ to make the mistakes of the combined classifier more evident, and train a classifier h_3 from the distribution, so that h_3 has correct classifications in $\mathcal{X}_2$ and $\mathcal{X}_3$. Then, by combining h_1, h_2 and h_3, we have a perfect classifier, since in each space of $\mathcal{X}_i$, $i = 1, 2, 3$, at least two classifiers make correct classifications. To conclude, boosting works by training a set of learners sequentially and combining them for prediction, where the later learners focus more on the mistakes of the earlier learners. See Algorithm (1).

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 1 Boosting procedure

Require: Input: sample distribution $\mathcal{D}$, base learning algorithm L, number of learning rounds N
Require: $\mathcal{D}_1 = \mathcal{D}$ initialise distribution
1: **for** $n = 1$ **to** N **do**
2: $h_n = L(\mathcal{D}_n)$ train a weak learner from distribution $\mathcal{D}_n$
3: $\epsilon_n = P_{x\sim\mathcal{D}_n}(h_n(x) \neq f(x))$ evaluate the error of h_n
4: $\mathcal{D}_{n+1} =$ Adjust-Distribution $(\mathcal{D}_n, \epsilon_n)$
5: **end for**
Require: Output: $H(x) =$ Combine-Outputs $(\{h_1(x), ..., h_n(x)\})$

Freund et al. [1997] proposed the AdaBoost to specify Adjust-Distribution and Combine-Outputs . See Algorithm (2).

Algorithm 2 AdaBoost algorithm

Require: Input: data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$, base learning algorithm L, number of learning rounds N
Require: $\mathcal{D}_1 = \frac{1}{m}$ initialise the weight distribution
1: **for** $n = 1$ **to** N **do**
2: $h_n = L(D, \mathcal{D}_n)$ train a classifier h_n from D under distribution $\mathcal{D}_n$
3: $\epsilon_n = P_{x\sim\mathcal{D}_n}(h_n(x) \neq f(x))$ evaluate the error of h_n
4: **if** $\epsilon_n > \frac{1}{2}$ **then**
5: break
6: **end if**
7: $\alpha_n = \frac{1}{2}\ln(\frac{1-\epsilon_n}{\epsilon_n})$ determine the weight of h_n
8: $\mathcal{D}_{n+1}(x) = \frac{\mathcal{D}_n(x)}{Z_n} e^{-\alpha_n f(x) h_n(x)}$ update the distribution, where Z_n is a normalisation factor which enables $\mathcal{D}_{n+1}$ to be a distribution
9: **end for**
Require: Output: $H(x) = \text{sgn}\big(\sum_{n=1}^{N} \alpha_n h_n(x)\big)$

Considering a binary classification on classes $\{-1, +1\}$, Friedman et al. [2000] modified the AdaBoost algorithm by minimising the exponential loss function

$$L_{exp}(h|\mathcal{D}) = E_{x\sim\mathcal{D}}[e^{-f(x)h(x)}]$$

using additive weighted combination of weak learners as

$$H(x) = \sum_{n=1}^{N} \alpha_n h_n(x)$$

Minimising the exponential loss by H, the partial derivative for every x is zero, that is, $\partial_{H(x)} e^{-f(x)H(x)} = 0$, so that

$$H(x) = \frac{1}{2}\ln\frac{P(f(x) = 1|x)}{P(f(x) = -1|x)}$$

As a result,

$$\text{sgn}(H(x)) = \arg\max_{y\in\{-1,1\}} P(f(x) = y|x)$$

which implies that $\text{sgn}(H(x))$ achieves the Bayes error rate. The H is obtained by iteratively generating h_n and α_n.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.1.2.3 Bagging

Bootstrap AGGregatING (called Bagging) is composed of bootstrap and aggregation (see Breiman [1996c]). Since the combination of independent base learners will lead to a dramatic decrease of errors, we want to get the base learners as independent as possible. Ideally, given a data set, we want to sample a number of non-overlapped data subsets and then train a base learner from each of the subsets. However, this is in general not possible due to limited data. One solution is to consider data sample manipulation, such as data perturbation. For instance, bagging adopts the bootstrap distribution (see Efron et al. [1993]) for generating different base learners. It applies bootstrap sampling to obtain the data subsets for training the base learners. Given a training data set containing m number of training examples, a sample of m training examples will be generated by sampling with replacement. By applying the process N times, N samples of m training examples are obtained. Then, from each sample a base learner can be trained by applying the base learning algorithm. The outputs of the base learners are aggregated by voting for classification and averaging for regression. If the base learners are able to output confidence values, weighted voting or weighted averaging are often used. For instance, when forecasting instance (feature vector), the algorithm feeds the instance to its base classifiers and collects all of their outputs, and then votes the labels and takes the winner label as the prediction, where ties are broken arbitrarily. See Algorithm (3).

Algorithm 3 Bagging procedure

Require: Input: data set $\{(x_1, y_1), ..., (x_m, y_m)\}$, base learning algorithm L, number of base learner N

1: **for** $n = 1$ **to** N **do**

2: $h_n = L(D, \mathcal{D}_{bs})$ $\mathcal{D}_{bs}$ is the bootstrap distribution

3: **end for**

Require: Output: $H(x) = \arg\max_{y \in \mathcal{Y}} \sum_{n=1}^{N} I(h_n(x) = y)$

Note bagging has a large variance reduction effect which is particularly effective with unstable base learners.

3.1.3 Random forest

Random forests (RF) or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees (see Ho [1995] [1998]). The algorithm was extended by Breiman [2001]. Another extension combines the bagging idea from Breiman and random selection of features, introduced first by Ho and later independently by Amit et al. [1997], in order to construct a collection of decision trees with controlled variance.

The idea of ensemble learning is that a single classifier is not sufficient for determining class of test data since the latter is not able to distinguish between noise and pattern. Thus, random decision forests correct for decision trees' habit of overfitting to their training set. Ho established that forests of trees splitting with oblique hyperplanes can gain accuracy as they grow without suffering from overtraining, as long as the forests are randomly restricted to be sensitive to only selected feature dimensions. He eventually concluded that other splitting methods behave similarly, as long as they are randomly forced to be insensitive to some feature dimensions. Note that this observation of a more complex classifier (a larger forest) getting more accurate nearly monotonically is in sharp contrast to the common belief that the complexity of a classifier can only grow to a certain level of accuracy before being hurt by overfitting.

3.1.3.1 Tree bagging

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, ..., x_m$ with responses $Y = y_1, ..., y_m$, bagging repeatedly (N times) selects a random sample with replacement of the training set and fits trees to these samples.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 4 Tree Bagging

1: **for** $n = 1$ **to** N **do**
2: sample, with replacement, n training examples from X, Y: called X_n, Y_n
3: train a classification or regression tree f_n on X_n, Y_n
4: **end for**

After training, predictions for unseen samples $x^{'}$ can be made by averaging the predictions from all the individual regression trees on $x^{'}$

$$\hat{f} = \frac{1}{N} \sum_{n=1}^{N} f_n(x^{'})$$

or by taking the majority vote in the case of classification trees.
This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Bootstrap sampling is a way of de-correlating the trees by showing them different training sets. An estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on $x^{'}$

$$\sigma^2 = \frac{1}{N-1} \sum_{n=1}^{N} \left(f_n(x^{'}) - \hat{f}\right)^2$$

An optimal number of trees N can be found using cross-validation, or by observing the out-of-bag error.

3.1.3.2 The procedure

Random forests (RF) is an extension of bagging incorporating randomised feature selection. During the construction of a component decision tree, at each step of split selection (deterministic), RF first randomly selects a subset of features, and then carries out the conventional split selection procedure within the selected feature subset. That is, RF use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. It is called feature bagging. Typically, for a classification problem with p features, $\sqrt{p}$ features are used in each split.
The parameter K controls the incorporation of randomness. When K equals the total number of features, the constructed decision tree is identical to the traditional deterministic decision tree, but when $K = 1$, a feature will be selected randomly. Breiman [2001] suggested $K = \log p$, the logarithm of the number of features.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 5 Random tree algorithm

Require: Input: data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$, feature subset size K
$N \leftarrow$ create a tree node based on D
if all instances in the same class **then**
  return N
end if
$\mathcal{F} \leftarrow$ the set of features that can be split further
if $\mathcal{F}$ is empty **then**
  return N
end if
$\widetilde{\mathcal{F}} \leftarrow$ select K features from $\mathcal{F}$ randomly
$N \cdot f \leftarrow$ the feature that has the best split point in $\widetilde{\mathcal{F}}$
$N \cdot p \leftarrow$ the best split point on $N \cdot f$
$D_l \leftarrow$ subset of D with values of $N \cdot f$ smaller than $N \cdot p$
$D_r \leftarrow$ subset of D with values of $N \cdot f$ no smaller than $N \cdot p$
$N_l \leftarrow$ call the process with parameters (D_l, K)
$N_r \leftarrow$ call the process with parameters (D_r, K)
return N
Require: Output: a random decision tree

Adding one further step of randomisation yields extremely randomized trees, or ExtraTrees. These are trained using bagging and the random subspace method, like in an ordinary random forest, but additionally the top-down splitting in the tree learner is randomised. Instead of computing the locally optimal feature/split combination (based on, information gain or the Gini impurity), for each feature under consideration, a random value is selected for the split. This value is selected from the feature's empirical range (in the tree's training set, that is, the bootstrap sample).
The technique is as follows: The first step in measuring the variable importance in a data set $D_n = \{(X_i, Y_i)\}_{i=1}^n$ is to fit a random forest to the data. During the fitting process the out-of-bag error for each data point is recorded and averaged over the forest (errors on an independent test set can be substituted if bagging is not used during training). To measure the importance of the jth feature after training, the values of the jth feature are permuted among the training data and the out-of-bag error is again computed on this perturbed data set. The importance score for the jth feature is computed by averaging the difference in out-of-bag error before and after the permutation over all trees. The score is normalised by the standard deviation of these differences.
As an example, we consider a random forest that performs sampling with replacement, such that, given N trees that are to be learnt, the results are based on these data set samples. Each tree is learnt using 3 features selected randomly. After the creation of n trees, when testing data is used, the decision which majority of trees come up with is considered as the final output. This approach also avoids problem of overfitting. The implementation is summarised in the Algorithm (6).

Algorithm 6 Random Forest

Require: Input: training set D, number of trees in the ensemble N
Require: Output: a composite model M_*
for $n = 1$ **to** N **do**
  create bootstrap sample D_n by sampling D with replacement
  select three features randomly
  use D_n and randomly select three features to derive tree M_n
end for
return M_*

Electronic copy available at: https://ssrn.com/abstract=3307566

3.1.4 Towards combination methods

Ensemble methods resort to combination to achieve a strong generalisation ability, where the combination method plays a crucial role. Some of the reasons for improvement are

- Statistical issue: high variance
- Computational issue: high computational variance
- Representational issue: high bias

Through combination, the variance as well as the bias of learning algorithms may be reduced.

3.1.4.1 Averaging

Averaging is the most fundamental combination method for numeric outputs. To illustrate the method, we take regression as an example. Thus, we consider the regression setting and suppose that we estimate N models (learners) $\{\mathcal{M}_1, ..., \mathcal{M}_N\}$, with $h_i = \mathcal{M}_i$, $i = 1, ..., N$, on a single data set D leading to corresponding predictions $h_1(x), ..., h_N(x)$ with instance x (or feature vector) where $h_i(x) = \widehat{f}_i(x) \in \mathbb{R}$. In model averaging, we compute the prediction by directly averaging the outputs of individual learners

$$H(x) = \widehat{f}_{ave}(x) = \sum_{i=1}^{N} \omega_i h_i(x) \tag{3.1.1}$$

where ω_i, $i = 1, .., N$, are fixed model weights. There are several ways for choosing the weights ω_i, $i = 1, .., N$.

3.1.4.1.1 Simple averaging The simplest one being to set $\omega_i = \frac{1}{N}$ to get a simple model average. Suppose the underlying true function we try to learn is $f(x)$, and x is sampled according to a distribution $p(x)$. The output of each learner can be written as the true value plus an error item as follows:

$$h_i(x) = f(x) + \epsilon_i(x) \, , i = 1, ..., N$$

Then, the mean squared error of h_i can be written as

$$\mathcal{E}(h) = \int \big(h_i(x) - f(x)\big)^2 p(x)dx = \int \epsilon_i^2(x)p(x)dx$$

and the averaged error made by the individual learners is

$$\overline{\mathcal{E}}(h) = \frac{1}{N} \sum_{i=1}^{N} \int \epsilon_i^2(x)p(x)dx$$

Similarly, the expected error of the combined learner is

$$\mathcal{E}(H) = \int \Big(\frac{1}{N} \sum_{i=1}^{N} h_i(x) - f(x)\Big)^2 p(x)dx = \int \big(\frac{1}{N} \sum_{i=1}^{N} \epsilon_i(x)\big)^2 p(x)dx$$

Thus, one can see that

$$\mathcal{E}(H) \leqslant \overline{\mathcal{E}}(h)$$

which states that the expected ensemble error will be no larger than the averaged error of the individual learners. In addition, if we assume that the errors ϵ_i, $i = 1, ..., N$, have zero mean and are uncorrelated

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\int \epsilon_i p(x)dx = 0 \text{ and } \int \epsilon_i \epsilon_j p(x)dx = 0 \text{ for } i \neq j$$

we get

$$\mathcal{E}(H) = \frac{1}{N}\overline{\mathcal{E}}(h)$$

so that the ensemble error is smaller by a factor of N than the averaged error of the individual learners.

***Remark* 3.1.1** *In general, the errors are typically highly correlated since the individual learners are trained on the same problem.*

3.1.4.1.2 Weighted averaging More generally, assume $\omega_i \geqslant 0$ and $\sum_{i=1}^N \omega_i = 1$, then

$$\begin{aligned}
\mathcal{E}(H) &= \int \Big(\sum_{i=1}^N \omega_i h_i(x) - f(x)\Big)^2 p(x)dx \\
&= \int \Big(\sum_{i=1}^N \omega_i h_i(x) - f(x)\Big)\Big(\sum_{j=1}^N \omega_j h_j(x) - f(x)\Big)p(x)dx \\
&= \sum_{i=1}^N \sum_{j=1}^N \omega_i \omega_j C_{ij}
\end{aligned}$$

where

$$C_{ij} = \int \big(h_i(x) - f(x)\big)\big(h_j(x) - f(x)\big)p(x)dx$$

and C is the correlation matrix. Thus, the optimal weights can be solved by

$$\omega = \arg\min_{\omega} \sum_{i=1}^N \sum_{j=1}^N \omega_i \omega_j C_{ij}$$

Applying the Lagrange multiplier method, and assuming the correlation matrix C to be invertible, we get

$$\omega_i = \frac{\sum_{j=1}^N C_{ij}^{-1}}{\sum_{k=1}^N \sum_{j=1}^N C_{kj}^{-1}}$$

which provides a closed-form solution to the optimal weights.

***Remark* 3.1.2** *The correlation matrix is usually singular or ill-conditioned, since the errors of the individual learners are typically highly correlated and many individual learners may be similar since they are trained on the same problem.*

Therefore ω_i is generally infeasible, and moreover, it does not guarantee non-negative solutions.
Note, voting, is special cases or variants of weighted averaging. More generally any ensemble method can be regarded as trying a specific way to decide the weights for combining the individual learners, and different ensemble methods can be regarded as different implementations of weighted averaging.
For example, given a training set with m number of training examples, we can choose the coefficients by least squares

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\min_{\omega_i : i \in [1,N]} \sum_{k=1}^{m} \left(y_k - \hat{f}_{ave}(x_k) \right)^2$$

where we may want to impose restrictions on the weights such as non-negativity and a total sum of one. However, the least squares estimation might not work well since the regression models $\{h_1(\cdot), ..., h_N(\cdot)\}$ were previously estimated based on the same training data.
The minimisation does not take into account the complexity of the individual models, and will tend to overfit by putting too much weight on the most complex models (which will have low training errors).

***Remark* 3.1.3** *When the data is noisy and insufficient, the estimated weights are often unreliable.*

In general, simple averaging is appropriate for combining learners with similar performances, whereas if the individual learners exhibit non-identical strength, weighted averaging with unequal weights may achieve a better performance.

3.1.4.2 Voting

Voting is the most popular and fundamental combination method for nominal outputs. Thus, to illustrate the method, we consider classification and suppose given a set of N individual classifiers $\{h_1, ..., h_N\}$. We want to combine the h_i to predict the class label from a set of l possible class labels $\{c_1, ..., c_l\}$. We assume that for any instance x, the outputs of the classifier h_i are given as an l-dimensional label vector $(h_{i,1}(x), ..., h_{i,l}(x))^\top$, where $h_{i,j}(x)$ is the output of h_i for the class label c_j. The output can take different types of values according to the information provided by the individual classifiers. For instance:

- Crisp label: $h_{i,j}(x) \in \{0, 1\}$ which takes value one if h_i predicts c_j as the class label and zero otherwise.
- Class probability: $h_{i,j}(x) \in [0, 1]$ which is an estimate of the posterior probability $P(c_j|x)$.

For classifiers that produce un-normalised margins, such as SVMs, calibration methods such as Platt scaling or Isotonic Regression can be used to convert such an output to a probability.

3.1.4.2.1 Majority voting Majority voting (MV) is the most popular voting method. Every classifier votes for one class label, and the final output class label is the one that receives more than half of the votes; if none of the class labels receives more than half of the votes, a rejection option will be given and the combined classifier makes no prediction. The output class label of the ensemble is

$$H(x) = \begin{cases} c_j \text{ if } \sum_{i=1}^{N} h_{i,j}(x) > \frac{1}{2} \sum_{k=1}^{l} \sum_{i=1}^{N} h_{i,k}(x) \\ \text{rejection if a tax credit} \end{cases}$$

For example, given a total of N classifiers for a binary classification problem, the ensemble decision will be correct if at least $\lfloor \frac{N}{2} + 1 \rfloor$ classifiers choose the correct class label.
Assume that the outputs of the classifiers are independent and each classifier has an accuracy p [2]. Then, the probability of the ensemble for making a correct decision can be calculated using a binomial distribution (see Hansen et al. [1990]). The probability of obtaining at least $\lfloor \frac{N}{2} + 1 \rfloor$ correct classifiers out of N is

$$P_{mv} = \sum_{k=\lfloor \frac{N}{2}+1 \rfloor}^{N} \binom{N}{k} p^k (1-p)^{N-k}$$

Lam et al. [1997] showed that

[2] each classifier makes a correct classification at probability p

Electronic copy available at: https://ssrn.com/abstract=3307566

- if $p > \frac{1}{2}$, then P_{mv} is monotonically increasing in N, and

$$\lim_{N\to\infty} P_{mv} = 1$$

- if $p < \frac{1}{2}$, then P_{mv} is monotonically decreasing in N, and

$$\lim_{N\to\infty} P_{mv} = 0$$

- if $p = \frac{1}{2}$ then $P_{mv} = \frac{1}{2}$ for any N

***Remark* 3.1.4** *This result is obtained based on the assumption that the individual classifiers are statistically independent, but in practice the classifiers are generally highly correlated since they are trained on the same problem.*

3.1.4.2.2 Plurality voting Plurality voting (PV) takes the class label which receives the largest number of votes as the final winner. That is, the output class label of the ensemble is

$$H(x) = c_{\arg\max_j \sum_{i=1}^N h_{i,j}(x)}$$

and ties are broken arbitrarily. Note, plurality voting does not have a reject option. Further, in the case of binary classification, plurality voting coincides with majority voting.

3.1.4.2.3 Weighted voting If the individual classifiers have unequal performance, we want to give more power to the stronger classifiers in voting; this is realised by weighted voting. The output class label of the ensemble is

$$H(x) = c_{\arg\max_j \sum_{i=1}^N \omega_i h_{i,j}(x)}$$

where ω_i is the weight assigned to the classifier h_i. Again, we want $\omega_i > 0$ and $\sum_{i=1}^N \omega_i = 1$. Further, we need to decide how to obtain the weights. We let $\ell = (\ell_1, ..., \ell_N)^\top$ be the outputs of the individual classifiers, where ℓ_i is the class label predicted for the instance x by the classifier h_i and p_i denotes the accuracy of h_i. There is a Bayesian optimal discriminant function for the combined output on class label c_j

$$H_j(x) = \log\left(P(c_j)P(\ell|c_j)\right)$$

Assuming that the outputs of the individual classifiers are conditionally independent

$$P(\ell|c_j) = \prod_{i=1}^N P(\ell_i|c_j)$$

then one can show

$$H_j(x) = \log\left(P(c_j) + \sum_{i=1,\ell_i=c_j}^N \log\frac{p_i}{1-p_i} + \sum_{i=1}^N \log(1-p_i)\right)$$

Since $\sum_{i=1}^N \log(1-p_i)$ does not depend on the class label c_j, and $\ell_i = c_j$ can be expressed by the result of $h_{i,j}(x)$, the discriminant function can be simplified to

$$H_j(x) = \log\left(P(c_j) + \sum_{i=1}^N h_{i,j}(x)\log\frac{p_i}{1-p_i}\right)$$

The first term on the RHS does not rely on the individual learners, while the second term discloses that the optimal weights for weighted voting satisfy

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\omega_i \propto \log \frac{p_i}{1-p_i}$$

which shows that the weights should be in proportion to the performance of the individual learners.

***Remark* 3.1.5** *The above formula obtained by assuming independence among the outputs of the individual classifiers, yet this does not hold since all the individual classifiers are trained on the same problem and they are usually highly correlated.*

3.1.4.2.4 Soft voting Soft voting (SV) is applied to classifiers which produce class probability outputs. The individual classifier h_i outputs a l-dimensional vector $(h_{i,1}(x), ..., h_{i,l}(x))^\top$ for the instance x, where $h_{i,j}(x) \in [0, 1]$ can be regarded as an estimate of the posterior probability $P(c_j|x)$. In the case of uniform weighting, the simple soft voting average all the individual outputs, getting the final output for class c_j as

$$H_j(x) = \frac{1}{N}\sum_{i=1}^{N} h_{i,j}(x)$$

On the other hand, the weighted soft voting method can be any of the following three forms:

1. A classifier-specific weight is assigned to each classifier, and the combined output for class c_j is

$$H_j(x) = \sum_{i=1}^{N} \omega_i h_{i,j}(x)$$

   where ω_i is the weight of the classifier h_i.

2. A class-specific weight is assigned to each classifier per class, and the combined output for class c_j is

$$H_j(x) = \sum_{i=1}^{N} \omega_{i,j} h_{i,j}(x)$$

   where $\omega_{i,j}$ is the weight of the classifier h_i for the class c_j.

3. A weight is assigned to each example of each class for each classifier, and the combined output for class c_j is

$$H_j(x) = \sum_{i=1}^{N}\sum_{k=1}^{m} \omega_{i,j,k} h_{i,j}(x)$$

   where $\omega_{i,j,k}$ is the weight of the instance x_k of the class c_j for the classifier h_i.

Focussing on case (2), we have

$$h_{i,j}(x) = P(c_j|x) + \epsilon_{i,j}(x)$$

where $\epsilon_{i,j}(x)$ is the approximation error. In classification, the target output is given as a class label. In the case of an unbiased estimation the combined output $H_j(x)$ is also unbiased, and we can obtain a variance-minimised unbiased estimation of $H_j(x)$ for $P(c_j|x)$ by estimating the weights. Minimizing the variance of the combined approximation error $\sum_{i=1}^{N} \omega_{i,j}\epsilon_{i,j}(x)$ under the constraints $\omega_{i,j} \geqslant 0$ and $\sum_{i=1}^{N} \omega_{i,j} = 1$, we get the optimisation problem

$$\omega_j = \arg\min_{\omega_j} \sum_{k=1}^{m} \Big(\sum_{i=1}^{N} \omega_{i,j} h_{i,j}(x) - I(f(x_k) = c_j)\Big)^2, \, j = 1, ..., l$$

and solve for the weights. In general, soft voting is used for homogeneous ensembles. In the case of heterogeneous ensembles, one must perform a calibration where the class probability outputs are converted to class label outputs by setting $h_{i,j}(x) = 1$ if $h_{i,j}(x) = \max\{h_{i,j}(x)\}$ and 0 otherwise. The voting methods for crisp labels can then be applied.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.2 Online learning and regret-minimising algorithms

3.2.1 Simple online algorithms

First, we introduce some notation which will be used through out this section. We use relint (S) to refer to the relative interior of a convex set S, which is the set S minus all of the points on the relative boundary. We use closure (S) to refer to the closure of S, the smallest closed set containing all of the limit points of S. For any subset S of $\mathbb{R}^d$, we let $\mathcal{H}(S)$ denote the convex hull of S. We let $\Delta_n = \{x \in \mathbb{R}^n : \sum_i^n x_i = 1, x_i \geqslant 0 \forall i\}$ be the n-dimensional probability simplex.

3.2.1.1 The Halving algorithm

To introduce online learning we are first going to present the Halving algorithm where a player has access to the prediction of N experts denoted by

$$f_{1,t}, ..., f_{N,t} \in \{0, 1\}$$

At each time $t = 1, .., T$, we observe $f_{i,t}$, i=1,..,N, and predict $p_t \in \{0, 1\}$. We then observe $y_t \in \{0, 1\}$ and suffer loss $I_{\{p_t \neq y_t\}}$. Suppose $\exists j$ such that $f_{j,t} = y_t$ for all $t \in [T]$. Then, the Halving theorem predicts $p_t = majority(C_t)$, where $C_1 = [N]$ and $C_t \subseteq [N]$ is defined below for $t > 1$.

Theorem 3.2.1 *If* $p_t = majority(C_t)$ *and*

$$C_{t+1} = \{i \in C_t : f_{i,t} = y_t\}$$

then we will make at most $\log_2 N$ *mistakes.*

In fact, for every t at which there is a mistake, at least half of the experts in C_t are wrong, so that $|C_{t+1}| \leqslant \frac{|C_t|}{2}$. It follows that $|C_T| \leqslant \frac{|C_1|}{2^M}$ where M is the tolal number of mistakes. Further, since there is a perfect expert, $|C_T| \geqslant 1$. As a result, recalling that $C_1 = [N]$, then $1 \leqslant \frac{N}{2^M}$, such that, after rearranging we get $M \leqslant \log_2 N$.

3.2.1.2 The weighted majority algorithm

To illustrate online, no-regret learning, we consider the problem of learning from expert advice where an algorithm must make a sequence of predictions based on the advice of a set of N experts and receive a corresponding sequence of losses. The aim of the algorithm being to achieve a cumulative loss that is almost as low as the cumulative loss of the best performing expert in hindsight. No statistical assumptions are made about these losses. At every time step $t \in \{1, .., T\}$, every expert $i \in \{1, .., N\}$ predict $f_{i,t} \in [0, 1]$ and receives a loss $l_{i,t} = l(f_{i,t}, y_t) \in [0, 1]$ such that the cumulative loss of expert i at time T is given by $L_{i,T} = \sum_{t=1}^T l_{i,t}$. We let the algorithm $\mathcal{A}$ (or player) maintains a weight $w_{i,t}$ for each expert i at time t, where $\sum_{i=1}^N w_{i,t} = 1$. Hence, these weights can be seen as a distribution over the experts. The algorithm then receives its own instantaneous loss

$$l_{\mathcal{A},t} = \sum_{i=1}^N w_{i,t} l_{i,t}$$

which can be interpreted as the expected loss the algorithm would receive if it always chose an expert to follow according to the current distribution. The cumulative loss of $\mathcal{A}$ up to time T is defined as

$$L_{\mathcal{A},T} = \sum_{t=1}^T l_{\mathcal{A},t} = \sum_{t=1}^T \sum_{i=1}^N w_{i,t} l_{i,t}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

For simplicity of exposition we use l_t, L_t, and w_t to refer to the vector of losses, vector of cumulative loss, and vector of weights, respectively, for each expert on round t. We will use the dot product to relate these vectors. Since the algorithm will not achieve a small cumulative loss if none of the experts perform well, we generally measure the performance of an algorithm in terms of its regret, defined as the difference between the cumulative loss of the algorithm and the loss of the best performing expert

$$R_T = L_{\mathcal{A},T} - \min_{i\in\{1,..,N\}} L_{i,T}$$

Hence, the algorithm is said to have no-regret if the average per time step regret approaches 0 as $T \to \infty$. The Randomized Weighted Majority (WM) algorithm (see Littlestone et al. [1994]) is an example of a no-regret algorithm, where the Weighted Majority uses weights

$$w_{i,t} = \frac{e^{-\eta L_{i,t-1}}}{\sum_{j=1}^N e^{-\eta L_{j,t-1}}}$$

where $\eta > 0$ is a learning rate parameter and the player's predict is $p_t = \sum_{i=1}^N w_{i,t} f_{i,t}$. The pseudo code for the weighted majority algorithm is

1. for $t = 1, .., t$ do
    - player observes $f_{1,t}, .., f_{N,t}$
    - player predicts p_t
    - adversary reveals outcome y_t
    - player suffers loss $l(p_t, y_t)$
    - experts suffer loss $l(f_{i,t}, y_t)$
2. end for

After T trials, the regret of WM can be bounded as

$$R_T = L_{WM(\eta),T} - \min_{i\in\{1,..,N\}} L_{i,T} \leqslant \eta T + \frac{\log N}{\eta}$$

such that if T is known in advance, setting $\eta = \sqrt{\frac{\log N}{T}}$ yields the standard $\mathcal{O}(\sqrt{T \log N})$ regret bound. Further, setting $\eta = \sqrt{8\frac{\log N}{T}}$ we get the bound

$$R_T \leqslant \sqrt{\frac{T}{2} \log N}$$

3.2.2 The online convex optimisation

3.2.2.1 The online linear optimisation problem

Given the probability simplex Δ_N, the pseudo code for the online linear optimisation is

1. for $t = 1, .., t$ do
    - player predicts $w_t \in \Delta_N$ (w_t is essentially a probability distribution)
    - adversary reveals $l_t \in \mathbb{R}^N$
    - player suffers loss $w_t.l_t$ where $l_{i,t} = l(f_{i,t}, y_t)$

Electronic copy available at: https://ssrn.com/abstract=3307566

2. end for

The learner suffers regret

$$R_T = \sum_{t=1}^{T} w_t \cdot l_t - \min_{w \in \Delta_N} \sum_{t=1}^{T} w \cdot l_t$$

where $l_t(w) = w \cdot l_t$. Note, for the simplex, the distribution w will place all the probability on the best expert. That is, the experts setting is just a special case of the online linear optimisation, where the set $\mathcal{K}$ is the N-simplex Δ_n.

It was proved that the weights chosen by WM are precisely those minimising a combination of empirical loss and an entropic regularisation term (see Kivinen et al. [1997]). That is, the weight vector w_t at time t is the solution to the following minimisation problem

$$\min_{w \in \Delta_N} w.L_{t-1} - \frac{1}{\eta} H(w)$$

where L_{t-1} is the vector of cumulative loss at time $t-1$, Δ_N is the probability simplex, and $H(\cdot)$ is the entropy function (see Section (2.1.2))

$$H(w) = -\sum_{i=1}^{N} w_i \log(w_i)$$

***Remark* 3.2.1** *The entropy function acts as a regularisation function for the weight vector w.*

3.2.2.2 Considering Bergmen divergence

Given the online linear optimisation problem described in Section (3.2.2.1), we consider the minimisation problem for the weight vector w_{t+1}

$$w_{t+1} \arg\min_{w \in \mathcal{K}} \eta \sum_{s=1}^{t} l_t(w) + R(w)$$

for some convex function $R(\cdot)$. We define $\Phi_0(w) = R(w)$ and $\Phi_t(w) = \Phi_{t-1}(w) + \eta l_t(w)$. We also let $D_f(x, y)$ be the Bergman divergence between x and y with respect to the function f (see details in Appendix (**??**)).

Lemma 3.2.1 *Suppose $\mathcal{K} = \mathbb{R}^N$, then for any $u \in \mathcal{K}$ we get*

$$\eta \sum_{t=1}^{T} [l_t(w) - l_t(u)] = D_{\Phi_0}(u, w_1) - D_{\Phi_T}(u, w_{T+1}) + \sum_{t=1}^{T} D_{\Phi_T}(w_t, w_{t+1})$$

and we also get

$$\sum_{t=1}^{T} l_t(w) \leqslant \inf_{u \in \mathcal{K}} \Big[\sum l_t(u) + \eta^{-1} D_R(u, w_1) \Big] + \eta \sum_{t=1}^{T} D_{\Phi_t}(w_t, w_{t+1})$$

Now, suppose that $\nabla R(w_1) = 0$ and $\mathbb{R}^N = \mathcal{K}$, then

$$w_{t+1} = \arg\min_{w \in \mathbb{R}^N} \big[\eta l_t(w) + D_{\Phi_{t-1}}(w, w_t) \big]$$

and the two minimisation problems are equivalent. That is,

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}\eta l_t(w) &= \Phi_t(w) - \Phi_{t-1}(w) \\ \eta l_t(w) + D_{\Phi_{t-1}}(w, w_t) &= \Phi_t(w) - \Phi_{t-1}(w) + D_{\Phi_{t-1}}(w, w_t)\end{aligned}$$

Assuming that the two equations are equivalent for $\tau \leqslant t$, and w minimises Φ_{t-1}

$$\begin{aligned}\nabla_w D_{\Phi_{t-1}}(w, w_t) &= \nabla_w \Phi_{t-1}(w, w_t) - \nabla_w \Phi_{t-1}(w_t) \\ \nabla \Phi_t(w_{t+1}) &= \nabla \Phi_{t-1}(w_t) = ... = \nabla \mathbb{R}(w_1) = 0\end{aligned}$$

thus, $w_{t+1} = \arg\min_{w \in \mathcal{K}} \Phi_t(w)$. Assuming l_t's are linear functions, and letting R^* be the Legendre dual of the function $R(\cdot)$ (see details in Appendix (**??**)), we get

***Corollary* 3** *1.* $\eta\big(\sum l_t.w_t - \sum l_t \cdot u\big) = D_R(u, w_1) + D_R(u, w_{t+1}) + \sum D_R(w_t, w_t - 1)$ *for any* $u \in \mathbb{R}^N$

2. $w_{t+1} = \nabla R^*(\nabla R(w_t) - \eta l_t)$

Recall, the online gradient descent is $w_{t+1} = w_t - \eta l_t$. Further, if $R = \frac{1}{2}\|\cdot\|^{\frac{1}{2}}$, $\nabla R(w) = w$, $\nabla R^*(w) = w$, and if $l_t(\cdot)$ are convex (but not necessary linear), then

Lemma 3.2.2 *If we choose* $w_{t+1} = \arg\min_{w \in \mathbb{R}^N} \big[\eta \nabla l_t(w)^\top w + D_R(w, w_t)\big]$
(or equivalently $w_{t+1} = \arg\min_{w \in \mathbb{R}^N} \eta \sum \big[\nabla l_t(w)^\top w + R(x)\big]$*), then*

$$\sum_{t=1}^{T} \big[l_t(w_t) - l_t(u)\big] \leqslant \eta^{-1} D_R(u, w_1) + \sum_{t=1}^{T} D_R(w_t, w_{t+1})$$

3.2.2.3 More on the online convex optimisation problem

The WM is an example of a broader class of algorithms collectively known as Follow the Regularized Leader (FTRL) algorithm (see Hazan et al. [2008]). The FTRL template can be applied to a wide class of learning problems falling under the general framework of Online Convex Optimisation (see Zinkevich [2003]). Other problems falling into this framework include online linear pattern classification, online Gaussian density estimation, and online portfolio selection. Further, the online linear optimisation problem is an extension of the expert setting where the weights w_t are chosen from a fixed bounded convex action space $\mathcal{K} \subset \mathbb{R}^N$ (see Rakhlin [2009]). The algorithm follows

1. Input: convex compact decision set $\mathcal{K} \subset \mathbb{R}^N$.
2. Input: strictly convex differentiable regularisation function $\mathcal{R}(\cdot)$ defined on $\mathcal{K}$.
3. Parameter: $\eta > 0$.
4. Initialise: $L_1 =< 0, .., 0 >$.
5. for $t = 1, .., T$ do
6. The learner selects action $w_t \in \mathcal{K}$ according to

$$w_t = \arg\min_{w \in \mathcal{K}} w.L_{t-1} + \frac{1}{\eta}\mathcal{R}(w)$$

7. Nature reveals l_t, learner suffers loss $l_t.w_t$

Electronic copy available at: https://ssrn.com/abstract=3307566

8. The learner updates $L_t = L_{t-1} + l_l$
9. end for

Abernethy et al. [2012] showed that the FTRL algorithms for online learning and the problem of pricing securities in a prediction market have a strong syntactic correspondence. To do so they needed to make two assumptions

***Assumption* 1** *For each time step t,* $\|l_t\| \leqslant 1$.

***Assumption* 2** *The regulasier* $\mathcal{R}(\cdot)$ *has the Legendre property (see Cesa-Bianchi et al. [2006]).* $\mathcal{R}$ *is strictly convex on relint* $(\mathcal{K})$ *and* $||\nabla\mathcal{R}(w)|| \to \infty$ *as* $w \to$ *relint* $(\mathcal{K})$.

As a result, the solution to the above algorithm will always occur in the relative interior of $\mathcal{K}$ such that the optimisation is unconstrained.

3.3 Combining by learning

3.3.1 Stacking

The main idea of stacking is that we want to learn whether training data have been properly learned. Thus, a learner is trained to combine the individual learners. The individual learners are called the first-level learners, while the combiner is called the second-level learner, or meta-learner. Given the first-level learners we generate a new data set for training the second-level learner, where the outputs of the first-level learners are regarded as input features while the original labels are still regarded as labels of the new training data.
There are two approaches for combining models: voting and stacking. The differences between the stacking and voting framework are as follow:

- In contrast to stacking, no learning takes place at the meta-level when combining classifiers by a voting scheme.
- Label that is most often assigned to a particular instance is chosen as the correct prediction when using voting.

Practically, stacking is concerned with combining multiple classifiers generated by different learning algorithms $L_1, ..., L_N$ on a single dataset D, which is composed by a feature vector $d_i = (x_i, y_i)$. The stacking process can be broken into two phases:

1. Generate a set of base-level classifiers $h_1, ..., h_N$ where $h_n = L_n(D)$.
2. Train a meta-level classifier to combine the outputs of the base-level classifiers.

The implementation is summarised in the Algorithm (7).

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 7 Stacking procedure

Require: Input: data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$, first level learning algorithms $L_1, ..., L_N$, second level learning algorithm L

Require: Output: a composite model M_*

for $n = 1$ **to** N **do**
  $h_n = L_n(D)$ train a first-level learner by applying the first-level learning algorithm L_n
end for
$D^{'} = \varnothing$ generate a new data set
for $i = 1$ **to** m **do**
  for $n = 1$ **to** N **do**
    $z_{i,n} = h_n(x_i)$ train a first-level learner by applying the first-level learning algorithm L_n
  end for
  $D^{'} = D^{'} \cup \big((z_{i,1}, ..., z_{i,N}), y_i\big)$
end for
$h^{'} = L(D^{'})$ train the second-level learner $h^{'}$ by applying the second-level learning algorithm L to the new data set $D^{'}$

Require: Output: $H(x) = h^{'}\big(h_1(x), ..., h_N(x)\big)$

To avoid overfitting, it is suggested that the instances used for generating the new data set are excluded from the training examples for the first-level learners, and a cross validation or leave-one-out procedure is often recommended. The stacking framework is as follows:

- The training set for the meta-level classifier is generated through a leave-one-out cross validation process

$$\forall i = 1, ..., m \text{ and } \forall n = 1, ..., N \text{ then } h_n^i = L_n(D - d_i)$$

- The learned classifiers are then used to generate predictions for $d_i : z_i^n = h_n^i(x)$
- The meta-level dataset consists of examples of the form $\Big(\big(z_i^1, ..., z_i^N\big), y_i\Big)$, where the features are the predictions of the base-level classifiers and the class is the correct class of the example in hand.

In the case of k-fold cross-validation, the original training data set D is randomly split into k almost equal parts $D_1, ..., D_k$. Let D_j and $D_{(-j)} = D \backslash D_j$ to be the test and training sets for the jth fold. Given N learning algorithms, a first-level learner $h_n^{(-j)}$ is obtained by using the nth learning algorithm on $D_{(-j)}$. For x_i in D_j, we let $z_{i,n}$ denotes the output of the learner $h_n^{(-j)}$ on x_i. Then, at the end of the entire cross-validation process, the new data set is generated from the N individual learners as

$$D^{'} = \{\big(z_{i,1}, ..., z_{i,N}, y_i\big)\}_{i=1}^m$$

It is then applied to the second-level learning algorithm and the resulting learner $h^{'}$ is a function of $(z_1, ..., z_N)$ for y. Similarly to averaging described above, stacking is equivalent to solving the minimisation problem

$$\min_{\omega_m : m \in [1,M]} \sum_{i=1}^{N} \big(y_i - \widehat{f}_{avecv}(x_i)\big)^2 \tag{3.3.2}$$

where

$$\widehat{f}_{avecv}(x) = \sum_{m=1}^{M} \omega_m \widehat{f}_m^{(cv)}(x) \tag{3.3.3}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

and $\widehat{f}_m^{(cv)}(\cdot)$ are cross validation predictions. That is, model stacking fits a linear regression model based on constructed predictors derived from different models. We say that the linear regression model is therefore the meta model for the stack.

***Remark* 3.3.1** *More generally, we have*

$$\widehat{f}_{avecv}(x) = g\big(\widehat{f}_1^{(cv)}(x), ..., \widehat{f}_M^{(cv)}(x)\big)$$

There is no reason for $g(\cdot)$ to be linear, except for simplicity.

This idea can be generalised and used with any algorithm as a meta model. For instance, it can be applied to classification. Some options are:

- Hard voting classifier: (weighted) majority voting across several models.
- Soft voting classifier: (weighted) probability averages.
- Model stacking: fit a meta model such as a logistic regression using cross validation predictions from multiple models as inputs.

3.3.2 Infinite ensemble

Since most ensemble methods exploit only a small finite subset of hypotheses, Lin et al. [2008] developed an infinite ensemble framework that constructs ensembles with infinite hypotheses. It is based on support vector machines (SVM) and corresponds to learning the combination weights for all possible hypotheses. By embedding infinitely many hypotheses into a kernel, it can be found that the learning problem reduces to an SVM training problem with specific kernels.
Let $\mathcal{H} = \{h_\alpha : \alpha \in \mathcal{C}\}$ denote the hypothesis space, where $\mathcal{C}$ is a measure space. The kernel that embeds $\mathcal{H}$ is defined as

$$K_{\mathcal{H},r}(x_i, x_j) = \int_{\mathcal{C}} \Phi_{x_i}(\alpha)\Phi_{x_j}(\alpha)d\alpha$$

where $\Phi_x(\alpha) = r(\alpha)h_\alpha(x)$ and $r : \mathcal{C} \to \mathbb{R}_+$ is chosen such that the integral exists for all x_i, x_j. α denotes the parameter of the hypothesis h_α, and $Z(\alpha)$ means that Z depends on α. It has been proved that the kernel is valid. Following SVM, the framework formulates the following (primal) problem:

$$\begin{aligned}
&\min_{\omega \in L_2(\mathcal{C}), b\in\mathbb{R}, \xi\in\mathbb{R}^m} \frac{1}{2}\int_{\mathcal{C}} \omega^2(\alpha)d\alpha + C\sum_{i=1}^m \xi_i \\
&\text{s.t. } y_i\Big(\int_{\mathcal{C}} \omega(\alpha)r(\alpha)h_\alpha(x)d\alpha + b\Big) \geqslant 1 - \xi_i \\
&\xi_i \geqslant 0\ , \forall i = 1, ..., m
\end{aligned}$$

The final classifier obtained from this optimization problem is

$$g(x) = \text{sgn}\Big(\int_{\mathcal{C}} \omega(\alpha)r(\alpha)h_\alpha(x)d\alpha + b\Big)$$

By using the Lagrangian multiplier method and the kernel trick, the dual problem of the primal problem can be obtained, and the final classifier can be written in terms of the kernel $K_{\mathcal{H},r}$ as

Electronic copy available at: https://ssrn.com/abstract=3307566

$$g(x) = \text{sgn}\big(\sum_{i=1}^{m} y_i \lambda_i K_{\mathcal{H},r}(x_i, x) + b\big)$$

where the λ_i are the Lagrange multipliers. By equivalence, the above equation infinite ensemble over $\mathcal{H}$. The learning problem can be reduced to solving an SVM with the kernel $K_{\mathcal{H},r}$ and the final ensemble can be obtained by applying typical SVM solvers.

3.3.3 Other methods

There are many other combination methods available and we are going to briefly present a few of them.

3.3.3.1 The algebraic methods

The algebraic methods which is based on a probabilistic framework (see Kittler et al. [1998]). Again, we let $h_{i,j}(x)$ be the class probability of c_j output from h_i. Bayesian decision theory states that given N classifiers, the instance x should be assigned to the class c_j which maximises the posteriori probability $P(c_j|h_{1,j}, ..., h_{N,j})$. From Bayes theorem, it follows that

$$P(c_j|h_{1,j}, ..., h_{N,j}) = \frac{P(c_j)P(h_{1,j}, ..., h_{N,j}|c_j)}{\sum_{i=1}^{l} P(c_i)P(h_{1,j}, ..., h_{N,j}|c_i)}$$

where $P(h_{1,j}, ..., h_{N,j}|c_j)$ is the joint probability distribution of the outputs from the classifiers. Assuming that the outputs are conditionally independent, that is, $P(h_{1,j}, ..., h_{N,j}|c_j) = \prod_{i=1}^{N} P(h_{i,j}|c_j)$, it follows that

$$\begin{aligned} P(c_j|h_{1,j}, ..., h_{N,j}) &= \frac{P(c_j)\prod_{i=1}^{N} P(h_{i,j}|c_j)}{\sum_{i=1}^{l} P(c_i)\prod_{k=1}^{N} P(h_{k,j}|c_i)} \\ &= P^{N-1}(c_j)\prod_{i=1}^{N} P(c_j|h_{i,j}) \end{aligned}$$

Since $h_{i,j}$ is the probability output, we have $P(c_j|h_{i,j}) = h_{i,j}$. Thus, if all classes are with equal prior, we get the product rule for combination

$$H_j(x) = \prod_{i=1}^{N} h_{i,j}(x)$$

Following the same approach, Kittler et al. [1998] derived soft voting method, as well as the maximum/minimum/median rules which choose the maximum/minimum/median of the individual outputs as the combined output. For instance, the median rule generates the combined output according to

$$H_j(x) = \underset{i}{\text{med}}\big(h_{i,j}(x)\big)$$

where $\text{med}(\cdot)$ is the median statistic.

3.3.3.2 Mixture of experts

Mixture of experts (ME) works in a divide-and-conquer strategy where a complex task is broken up into several simpler and smaller subtasks, and individual learners (called experts) are trained for different subtasks (see Jacobs et al. [1991], Xu et al. [1995]). Gating is usually employed to combine the experts. The individual learners are generated for different subtasks and there is no need to devote to diversity (see Figure (3.2)). Thus, the key problem

Electronic copy available at: https://ssrn.com/abstract=3307566

becomes to find the natural division of the task and then derive the overall solution from sub-solutions. To do so, we make the experts local by targeting each expert to a distribution specified by the gating function, rather than the whole original training data distribution.

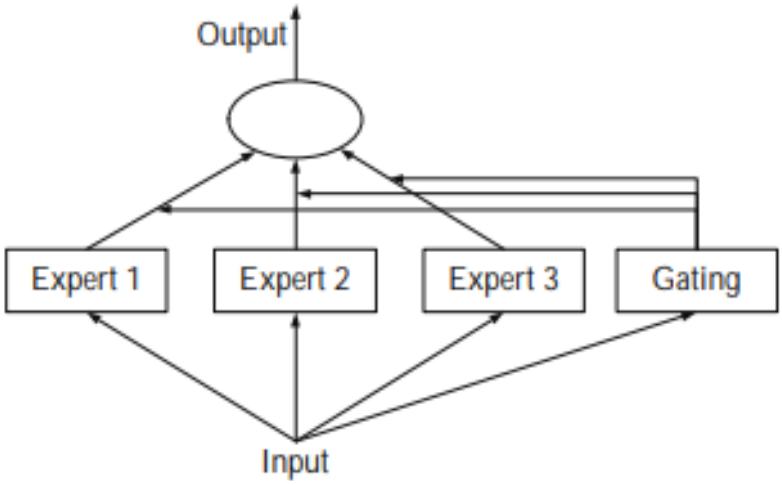


Figure 3.2: Mixture of experts

As an example, we assume N experts and let the output y be a discrete variable with possible values 0 and 1 for binary classification. Given an input x, each local expert h_i tries to approximate the distribution of y and obtains a local output $h_i(y|x;\theta_i)$ where θ_i is the parameter of the ith expert h_i. The gating function provides a set of coefficients $\pi_i(x;\alpha)$ weighting the contributions of experts where α is the parameter of the gating function. Thus, the final output of the ME is a weighted sum of all the local outputs produced by the experts given by

$$H(y|x;\Psi) = \sum_{i=1}^{N} \pi_i(x;\alpha) \cdot h_i(y|x;\theta_i)$$

where Ψ includes all unknown parameters. The output of the gating function is often modelled by the softmax function as

$$\pi_i(x;\alpha) = \frac{e^{v_i^\top x}}{\sum_{i=1}^{N} e^{v_j^\top x}}$$

where v_i is the weight vector of the ith expert in the gating function, and α contains all the elements in the vector v_i. When training, $\pi_i(x;\alpha)$ states the probability of the instance x appearing in the training set of the ith expert h_i. However, when testing, it defines the contribution of h_i to the final prediction.
The training procedure tries to achieve two goals: for given experts, to find the optimal gating function; for given gating function, to train the experts on the distribution specified by the gating function. The unknown parameters are usually estimated by the Expectation Maximisation (EM) algorithm (see Jordan et al. [1995], Xu et al. [1996]). Hierarchical mixture of experts (HME) extends mixture of experts (ME) into a tree structure where the experts are built from multiple levels of experts and gating functions (see Jordan et al. [1992]).

3.3.4 Diversity measures

A symmetric measure will keep the same when the values of 0 (incorrect) and 1 (correct) in binary classification are swapped.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.3.4.1 Pairwise measures

When measuring diversity, the simplest approach is to measure the pairwise similarity/dissimilarity between two learners, and then average all the pairwise measurements to compute the overall diversity. One advantage of pairwise measures is that they can be visualized in $2d$-plots.
Given the data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$, we consider binary classification and define the contingency table (see Table (3.1)) for two classifiers h_i and h_j, where $a + b + c + d = m$ are non-negative variables showing the numbers of examples satisfying the conditions specified by the corresponding rows and columns. We are going to present a few representative pairwise measures based on these variables.

Classifiers	$h_i = +1$	$h_i = -1$
$h_j = +1$	a	c
$h_j = -1$	b	d

Table 3.1: Contingency table

The disagreement measure (see Skalak [1996]) between h_i and h_j is defined as the proportion of examples on which two classifiers make different predictions

$$\underset{ij}{\text{dis}} = \frac{b+c}{m}$$

where dis_{ij} takes values in $[0, 1]$. The larger the value, the larger the diversity.
The Q-statistic (see Yule [1900]) of h_i and h_j is defined as

$$Q_{ij} = \frac{ad - bc}{ad + bc}$$

where Q_{ij} takes value in the range $[-1, 1]$. It is zero if h_i and h_j are independent, it is positive if h_i and h_j make similar predictions, otherwise it is negative.
The correlation coefficient (see Sneath et al. [1973]) of h_i and h_i is defined as

$$\rho_{ij} = \frac{ad - bc}{\sqrt{(a+b)(a+c)(c+d)(b+d)}}$$

which is classical when measuring the correlation between two binary vectors. Note, ρ_{ij} and Q_{ij} have the same sign, and $|\rho_{ij}| \geqslant |Q_{ij}|$.
The kappa-statistic (see Cohen [1960]) is defined as

$$\kappa_p = \frac{\Theta_1 - \Theta_2}{1 - \Theta_2}$$

where Θ_1 and Θ_2 are the probabilities that the two classifiers agree and agree by chance, respectively. The probabilities for h_i and h_j, on the data set D, are estimated as follows:

$$\begin{aligned} \Theta_1 &= \frac{a+d}{m} \\ \Theta_2 &= \frac{(a+b)(a+c) + (c+d)(b+d)}{m^2} \end{aligned}$$

Then, $\kappa_p = 1$ if the two classifiers totally agree on D, $\kappa_p = 0$ if they agree by chance, and $\kappa_p < 0$ occurs when the agreement is even less than what is expected by chance. All these measures do not require knowing the classification correctness.

Electronic copy available at: https://ssrn.com/abstract=3307566

3.3.4.2 Non-pairwise measures

We now focus on measuring the ensemble diversity directly rather than averaging pairwise measurements. We consider given a set of individual classifiers $\{h_1, ..., h_N\}$ and a data set $D = \{(x_1, y_1), ..., (x_m, y_m)\}$ where x_i is an instance and $y_i \in \{-1, +1\}$ is class label.
The Kohavi-Wolpert (KW) variance is based on the bias-variance decomposition of the error of a classifier (see Kohavi et al. [1996]). On an instance x, the variability of the predicted class label y is defined as

$$var_x = \frac{1}{2}\Big(1 - \sum_{y\in\{-1,+1\}} P^2(y|x)\Big)$$

Kuncheva et al. [2003] modified the variability to measure diversity by considering two classifier outputs: correct (denoted by $\tilde{y} = +1$) and incorrect (denoted by $\tilde{y} = -1$). They estimated $P(\tilde{y} = +1|x)$ and $P(\tilde{y} = -1|x)$ over individual classifiers as follows:

$$P(\tilde{y} = +1|x) = \frac{\rho(x)}{N} \text{ and } P(\tilde{y} = -1|x) = 1 - \frac{\rho(x)}{N}$$

where $\rho(x)$ is the number of individual classifiers that classify x correctly. By substituting these formulas in the KW-variance and averaging over the data set D the $\kappa\omega$ measure is obtained

$$\kappa\omega = \frac{1}{mN^2}\sum_{k=1}^{m}\rho(x_k)\big(N - \rho(x_k)\big)$$

where the larger the $\kappa\omega$ measurement, the larger the diversity.
Interrater agreement is a measure of interrater (interclassifier) reliability (see Fleiss [1981]). It can be used to measure the level of agreement within a set of classifiers (see Kuncheva et al. [2003]). It is measured as follows

$$\kappa = 1 - \frac{\frac{1}{N}\sum_{k=1}^{m}\rho(x_k)\big(N - \rho(x_k)\big)}{m(N-1)\overline{p}(1-\overline{p})}$$

where $\rho(x_k)$ is the number of classifiers that classify x_k correctly, and

$$\overline{p} = \frac{1}{mN}\sum_{i=1}^{N}\sum_{k=1}^{m} I(h_i(x_k) = y_k)$$

is the average accuracy of individual classifiers. Again, $\kappa = 1$ if the classifiers totally agree on D, and $\kappa \leqslant 0$ if the agreement is even less than what is expected by chance.
Kuncheva et al. [2003] showed that the KW-variance, the averaged disagreement dis_{avg} and the kappa-statistic κ were closely related. The relations are as follows

$$\kappa\omega = \frac{(N-1)}{2N}dis_{avg} \text{ and } \kappa = 1 - \frac{N}{(N-1)\overline{p}(1-\overline{p})}\kappa\omega$$

where $\overline{p}$ is as above.
In entropy, for an instance x_k, the disagreement will be maximised if a tie occurs in the votes of individual classifiers. Cunningham et al. [2000] directly calculated the Shannon's entropy on every instance and averaged them over D for measuring diversity, getting

$$Ent_{cc} = \frac{1}{m}\sum_{k=1}^{m}\sum_{y\in\{-1,+1\}} -P(y|x_k)\log P(y|x_k)$$

where

Electronic copy available at: https://ssrn.com/abstract=3307566

$$P(y|x_k) = \frac{1}{N}\sum_{i=1}^{N} I(h_i(x_k) = y)$$

can be estimated by the proportion of individual classifiers that predict y as the label of x_k. The computation of Ent_{cc} does not require to know the correctness of individual classifiers.
Assuming the correctness of the classifiers known, Shipp et al. [2002] defined their entropy measure as

$$Ent_{sk} = \frac{1}{m}\sum_{k=1}^{m} \frac{\min\big(\rho(x_k), N - \rho(x_k)\big)}{N - \lceil \frac{N}{2} \rceil}$$

where $\rho(x_k)$ is the number of individual classifiers that classify x correctly. The value of Ent_{sk} is in the range of $[0, 1]$, where 0 indicates no diversity and 1 indicates the largest diversity. Note, even though this formula does not use the logarithm function, it is easier to handle and faster to calculate.
Assuming that the diversity is maximised when the failure of one classifier is accompanied by the correct prediction of the other, Partridge et al. [1997] defined the generalised diversity as follows

$$gd = 1 - \frac{p(2)}{p(1)}$$

where

$$p(1) = \sum_{i=1}^{N} \frac{1}{N} p_i \text{ and } p(2) = \sum_{i=1}^{N} \frac{1}{N} \frac{(i-1)}{(N-1)} p_i$$

where p_i denotes the probability of i randomly chosen classifiers failing on a randomly drawn instance x. The gd value is in the range of $[0, 1]$, and the diversity is minimised when $gd = 0$. The authors also proposed a modified version of the generalised diversity, called coincident failure (CF), given by

$$cfd = \begin{cases} 0 \text{ if } p_0 = 1 \\ \frac{1}{1-p_0} \sum_{i=1}^{N} \frac{(N-i)}{(N-1)} p_i \text{ if } p_0 < 1 \end{cases}$$

In that setting, $cfd = 0$ if all classifiers give the same predictions simultaneously, and $cfd = 1$ if each classifier makes mistakes on unique instances.

Electronic copy available at: https://ssrn.com/abstract=3307566

Part II

Neural Networks

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 4

Introduction to artificial neural networks

While wavelet analysis enable to decompose complex systems into simpler elements (see Appendix (14.4)), in order to understand them, we can also gather simple elements to produce a complex system. Networks is one approach among others achieving that goal. They are characterised by a set of interconnected nodes seen as computational units receiving inputs and processing them to obtain an output. The connections between nodes, which can be unidirectional or bidirectional, determine the information flow between them. We obtain a global behaviour of the network, called emergent, since the abilities of the network supercede the one of its elements, making networks a very powerful tool. Since Neural Nets have been widely studied by computer scientists, electronic engineers, biologists, psychologists etc, they have been given many different names such as Artificial Neural Networks (ANNs), Connectionism or Connectionist Models, Multi-layer Percepetrons (MLPs), or Parallel Distributed Processing (PDP) to name a few. However, a small group of classic networks emerged as dominant such as Hopfield Networks (see Hopfield [1982]), Back Propagation (see Rumelhart et al. [1986] [1986b]), Competitive Networks and networks using Spiky Neurons.

Among the different networks existing, the artificial neural networks (ANNs) and the artificial recurrent neural networks (RNNs) are computational models designed by more or less detailed analogy with biological brain modules. They are inspired from natural neurons receiving signals through synapses located on the dendrites, or membrane of the neuron. When the signals received are strong enough (surpass a certain threshold), the neuron is activated and emits a signal through the axon, which might be sent to another synapse, and might activate other neurons. ANN is a high level abstraction of real neurons consisting of inputs (like synapses) multiplied by weights (strength of the respective signals), and computed by a mathematical function determining the activation of the neuron. Another function computes the output of the artificial neuron (sometimes in dependence of a certain threshold). Depending on the weights (which can be negative), the computation of the neuron will be different, such that by adjusting the weights of an artificial neuron we can obtain the output we want for specific inputs. In presence of a large number of neurons we must rely on an algorithm to adjust the weights of the ANN to get the desired output from the network. This process of adjusting the weights is called learning or training. The aim of training the network is to obtain the optimal network minimising the difference between the output and a target, given some input data. Then, the output of the optimal network becomes the optimal predictor. The optimisation process is usually done by gradient descent method, where the weights are updated by an amount being equal to the opposite of the gradient. This way of updating weights is called the standard back propagation.

4.1 The classic networks

Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions from examples. The basic architecture of an ANN is a network comprised of nodes (called *neurons* in biology) interacting with each other via incoming and outgoing weighted connections indicating the strength

Electronic copy available at: https://ssrn.com/abstract=3307566

degree of dendrite between neurons. While McCulloch et al. [1943] introduced the first ANN model, most modelling of neural learning networks has been based on synapses of a general type described by Hebb [1949] and Eccles [1953]. Given some input, the network will be activated and this activation signal will be passed throughout the rest of the network through the connections. Generally, an ANN model is specified by its topology, node characteristics and the training algorithms. There are a variety of ANNs with various topologies, node properties and training algorithms. Rosenblatt [1958] [1962] introduced the principles of perceptrons. Perceptrons were mainly modelled with neural connections in a forward direction $A \to B \to C \to D$. Consequently, the analysis of networks with strong backward coupling $A \leftrightarrows B \leftrightarrows C$ proved intractable. Further, perceptron modelling required synchronous neurons like a conventional digital computer. Hopfield [1982] overcame these difficulties and triggered the research interest in the area by demonstrating the computational capabilities of networks with symmetric connections. The main contribution of the paper was the introduction of the energy function of the network which turns to be their most useful property in the context of optimisation. To improve on the limitation of the perceptron, Fukushima [1980] proposed the multilayer percetron (MLP) also called feedforward network. Two of the most important distinct types of ANNs are feedforward and feedback network. ANNs with cycles are *feedback* networks, which are also referred to as *recurrent* neural networks (RNNs). Those networks with acyclic connections are called *feedforward* neural networks (FNNs). Algorithms such as back-propagation use gradient descent to tune network parameters to best fit a training set of input-output pairs, making the learning process robust to errors in the training data (see Rumelhart et al. [1986] [1986b]). Various successful applications to practical problems developed, such as learning to recognise handwritten characters (see LeCun et al. [1989]) and spoken words (see Lang [1990]), learning to detect fraudulent use of credit cards, drive autonomous vehicles on public highways (see Pomerleau [1993]). Rumelhart et al. [1994] provided a survey of practical applications. Since McCulloch et al. [1943] introduced the first ANN model, various models where developed with different functions, accepted values, topology, learning algorithms, hybrid models where each neuron has a larger set of properties etc. While there is various types of classifiers, neural network based classifiers dominate the literature. Yet, compared to traditional NNs, higher order neural networks (HONNs) have several unique characteristics, including

- stronger approximation with faster convergence property
- greater storage capacity
- higher fault tolerance capability

However, its major drawback is the exponential growth in the number of weights with the increasing order of the network. As a special case of the feedforward HONN, the Pi-Sigma networks (PSNs) introduced by Shin et al. [1991] have the capability of higher order neural networks, but using a smaller number of weights. In order to enhance the learning capability of neural network, many researchers have improved the system by combining other techniques such as fuzzy logic (see Zadeh [1994]), genetic algorithm (see Harrald et al. [1997]).

For simplicity of exposition we are first going to present the Hopfield networks (see Hopfield [1982]) as well as the multilayer percetron (MLP), and then briefly describe the back-propagation algorithm proposed by Rumelhart et al. [1986].

4.1.1 The Hopfield networks

4.1.1.1 Description

The Hopfield neural network model (see Hopfield [1982]) consists of a fully connected network of units (or neurons). The connections between the units are weighted, such that for any two units i and j, w_{ij} is the weight of the connection between them. The model assumes symmetrical weights ($w_{ij} = w_{ji}$) and in most of the cases zero self-coupling terms ($w_{ii} = 0$). A connection with positive weight is an excitatory connection, while a connection with negative weight is an inhibitory connection. A unit i is characterised by its output (or state) v_i, the activation (or network input)

Electronic copy available at: https://ssrn.com/abstract=3307566

u_i that receives from the other units, and a threshold θ_i. The network state is given by the output (or state) vector $v = \big(v_1, v_2, ..., v_n\big)$.
Each unit receives input from all the other units and forwards its output to all the other units. The output of a unit is updated by the dynamics of the network. In general, the output behaviour is described as a function of the activation, where the activation depends on the weighted summation of the inputs and the threshold. In general, we follow the dynamics rule from McCulloch et al. [1943], given by

$$v_i(t+1) = \Phi(u_i(t)) = \Phi\big(\sum_{j=1}^{n} w_{ij} v_j(t) + \theta_i\big) \tag{4.1.1}$$

where the activation is

$$u_i(t) = w_i \cdot v(t) + \theta_i = \sum_{j=1}^{n} w_{ij} v_j(t) + \theta_i \tag{4.1.2}$$

and θ_i is the threshold. The activation function $\Phi(x)$ can take several forms:

- it can be the unit step function (Heaviside function)

$$\Theta(x) = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ if } x \leqslant 0 \end{cases}$$

  in which case we have a discrete Hopfield network with binary states $\{0, 1\}$.

- the sign function

$$\text{sgn}(x) = \begin{cases} 1 \text{ if } x \geqslant 0 \\ -1 \text{ if } x < 0 \end{cases}$$

  in which case the network is discrete but with values $\{1, -1\}$.

- a network with continuous-valued units (analog Hopfield network), where the values fall in the range $[0, 1]$ or $[-1, 1]$ can be obtained with the sigmoid (or logistic) function

$$g_\beta(x) = \frac{1}{1 + e^{-2\beta x}}$$

  The parameter β is usually taken as $\beta = \frac{1}{T}$ where T is a virtual temperature. The latter adjusts the sharpness of the sigmoid (or hyperbolic tangent) function and at the limit $T \to 0$ (absolute temperature) the output becomes discrete. We can also use the hyperbolic tangent function

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Note that

$$\text{sgn}(x) = 2\Theta(x) - 1 \text{ and } tanh(\beta x) = 2g_\beta(x) - 1$$

so that the $\{0, 1\}$ (or $[0, 1]$) model can be easily transformed to $\{-1, 1\}$ (or $[-1, 1]$) and vice-versa. In the limit $\beta \to \infty$ the values become discrete.
All the networks above are deterministic in the sense that the next state is an explicit function of the previous state and the characteristics of the network. In the case of the stochastic Hopfield networks the probability of a unit to be in a particular state is drawn from a probability distribution such as Boltzmann or Cauchy. Alternatively, the stochastic network can be viewed as a deterministic network, where the threshold of a unit is variable and is drawn from a probability density.
Note, the updating policy can be

Electronic copy available at: https://ssrn.com/abstract=3307566

- synchronous, in which case all the units are updated simultaneously at each time step, or
- asynchronous, where either the units are updated in sequence one at each time step (sequential asynchronous update) or at each time step one randomly chosen unit is updated (random asynchronous update).

In summary, the properties that characterize different Hopfield networks are

- Discrete vs Continuous-Valued Units
- Deterministic vs Stochastic Networks
- Synchronous vs Asynchronous Update (Sequential or Random)

4.1.1.2 The energy function

The essential ingredient of the learning process is the Hebbian property which is the modification of w_{ij} by correlations like

$$\Delta w_{ij} = \big[v_i(t)v_j(t)\big]_{average}$$

where the average is some appropriate calculation over past history. In addition, the Hopfield model has stable limit points. That is, the energy function of a Hopfield network is a function defined over the state space of the network given by

$$\begin{aligned} E(v) &= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} v_i v_j w_{ij} - \sum_{i=1}^{n} v_i\theta_i \\ &= -\sum_{i=1}^{n} v_i\big(\frac{1}{2}\sum_{j=1}^{n} v_j w_{ij} + \theta_i\big) \\ &= -\sum_{i=1}^{n} v_i u_i = -u \cdot v \end{aligned} \tag{4.1.3}$$

Note, this function always decreases (not necessarily monotonically because of the threshold) during the evolution of the network due to the dynamics of the systems given in Equation (4.1.1). State changes (altering v_i) will continue until a least (local) E is reached. This case is isomorphic with an Ising model.
Thus, the energy function can be viewed as defining an energy landscape and the dynamics can be thought of as the motion of a small sphere on the energy surface under the influence of gravity and friction. Consequently, the network performs as a minimiser of the energy function and will reach a local or global minimum of the function during its evolution.
One can show that whenever a unit i changes state, the energy difference ΔE_i is given by

$$\Delta E_i = -u_i\Delta v_i = -\big(\sum_{j=1}^{n} w_{ij}v_j + \theta_i\big)\Delta v_i \tag{4.1.4}$$

where u_i is the activation in Equation (4.1.2). As an example, we consider the $\{0,1\}$ network with dynamics given in Equation (4.1.1). Then

- If the activation is positive, then either $\Delta u_i = 0$ $(1 \to 1)$ or $\Delta u_i = 1$ $(0 \to 1)$ and thus $\Delta E_i \leqslant 0$.
- If the activation is negative, then either $\Delta u_i = 0$ $(0 \to 0)$ or $\Delta u_i = -1$ $(1 \to 0)$ and thus $\Delta E_i \leqslant 0$.

The argument is similar for the other models. Note, the main property of the energy function given in Equation (4.1.3) is held only if the weights of the connections are symmetric and the self-coupling terms are zero or positive. This is the case for the Hopfield network so that its energy function is called Lyapunov function.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.1.2 The multilayer perceptron

4.1.2.1 Description

The *multilayer perceptron* (MLP) proposed by Fukushima [1980] and popularised by Rumelhart et al. [1986b] is formed in layers, the idea being that multilayer ANN can approximate any continuous function. This algorithm is a layer feed-forward ANN, since the artificial neurons are organised in layers with signals sent forward and with errors propagated backwards. Hence the MLP is also called feedforward network. The network receives inputs via neurons in the input layer, and the output of the network is given via neurons on the output layers. There may be one or more intermediate hidden layers. For clarity, we consider a network with three layers, that is, an input layer, a hidden layer, and an output layer. The hidden layer is used for capturing the non-linear relationships among variables. It is illustrated in Figure 4.1, where the bottom layer is the *input layer*. Then the information is propagated to the middle layer, called the *hidden layer*. Finally, the output layer receives the incoming value. This process is termed the *forward pass*. Note that the S-shaped curve is the squashing function which forces the output of the unit to fall between certain bounded interval. Except for the input layer, the output value of each unit in the other layers should be passed through the squashing function before transferring to the next layer. The back-propagation algorithm uses supervised learning where we provide the algorithm with examples of the inputs and outputs we want the network to compute, and then the error (difference between actual and expected results) is calculated. Defining the network function f_{out} as a particular implementation of a composite function from input to output space, the learning problem consists in finding the optimal combination of weights so that f_{out} approximates a given function f as closely as possible (see Section (8.1.1)). However, in practice the function f is not given explicitly but only implicitly through some examples.

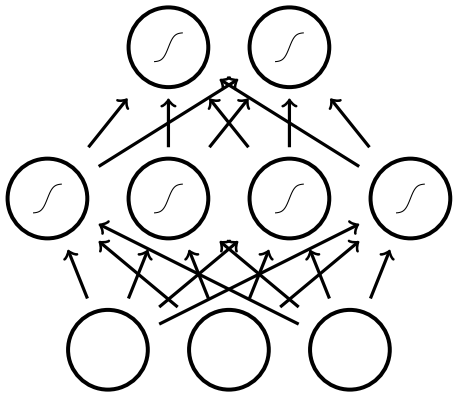


Figure 4.1: **A multilayer perceptron.** The three layers are input, hidden, and output layers, respectively, from bottom to top. The S-shaped curves in the hidden and output layers are the sigmoid squashing functions applying to the net values.

4.1.2.2 The universal approximation theorem

Cybenko [1989] hinted at the universal approximation theorem for sigmoid activation functions. Funahashi [1989] and Hornik et al. [1989] showed that a MLP with one hidden layer and sufficient non-linear units could approximate any continuous function on a compact input domain to arbitrary precision. Therefore, MLP are recognised as universal approximators. Hornik [1991] showed that it is not the specific choice of the activation function, but rather the multilayer feedforward architecture itself which gives neural networks the potential of being universal approximators. The universal approximation theorem can be stated as follows:

Theorem 4.1.1 *The universal approximation theorem*
Let $\Phi(\cdot)$ be a nonconstant, bounded, and continuous function. Let I_m denotes the m-dimensional unit hypercube

Electronic copy available at: https://ssrn.com/abstract=3307566

$[0,1]^m$. *The space of continuous functions on* I_m *is denoted by* $C(I_m)$. *Then, given any* $\epsilon > 0$ *and any function* $f \in C(I_m)$, *there exist an integer* N, *real constants* $v_i, b_i \in \mathbb{R}$ *and real vectors* $w_i \in \mathbb{R}^m$, *where* $i = 1, ..., N$, *such that we may define:*

$$F(x) = \sum_{i=1}^{N} v_i \Phi(w_i^\top x + b_i)$$

as an approximate realisation of the function f *where* f *is independent of* Φ. *That is,*

$$|F(x) - f(x)| < \epsilon$$

for all $x \in I_m$. *In other words, functions of the form* $F(x)$ *are dense in* $C(I_m)$.

Note, the theorem still holds when replacing I_m with any compact subset of $\mathbb{R}^m$. Kurkova [1992] took advantage of techniques developed by Kolmogorov to give a direct proof of the universal approximation capabilities of perceptron type networks with two hidden layers. Telgarsky [2016] argued that the theorem was only true in the case where the hidden layers were exponentially large.

4.1.3 Gradient descent and the delta rule

As explained by Rumelhart et al. [1986] [1986b], one way forward to finding the optimal combination of weights in the multilayer perceptron is to consider the delta rule, which uses gradient descent to search the hypothesis space of possible weight vectors to find the weights best fitting the training examples. The gradient descent provides the basis for the backpropagation algorithm, which can learn networks with many interconnected units.
Given a vector of input $x \in \mathbb{R}^n$ together with a weight vector $w \in \mathbb{R}^n$, we consider the task of training an unthresholded perceptron, that is, a linear unit for which the output O is given by

$$O(x) = w \cdot x$$

A linear unit is the first stage of a perceptron without the threshold. We must specify a measure for the training error of a hypothesis (weight vector), relative to the set $\mathcal{D}$ of training examples. A common measure is to use

$$E(w) = \frac{1}{2} \sum_{p=1}^{P} (O_p - d_p)^2$$

where d_p is the target output for training example p and O_p is the output of the linear unit for training example p. One can show that under certain conditions, the hypothesis minimising E is also the most probable hypothesis in H given the training data. The entire hypothesis space of possible weight vectors and their associated E values produce an error surface. Given the way in which we defined E, for linear units, this error surface must always be parabolic with a single global minimum.
The gradient descent search (see Section (10.2.1.1)) determines a weight vector minimising E by starting with an arbitrary initial weight vector, and then repeatedly modifying it in small steps. At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface, until a global minimum error is reached. This direction is found by computing the derivative of E with respect to each component of the vector w, called the gradient of E with respect to w, given by

$$\nabla E(w) = \left[\frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n}\right]$$

The gradient specifies the direction that produces the steepest increase in E, and its negative produces the steepest decrease. Hence, the training rule for gradient descent is

$$w \leftarrow w + \Delta w$$

Electronic copy available at: https://ssrn.com/abstract=3307566

where

$$\Delta w = -\eta \nabla E(w)$$

where $\eta > 0$ is the learning rate determining the step size in the gradient descent search. The negative sign implies that we move the weight vector in the direction that decreases E. Given the definition of $E(w)$ we can easily differentiate the vector of $\frac{\partial E}{\partial w_i}$ derivatives as

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{p=1}^{P} (O_p - d_p)^2 \ , i = 1, ..., n$$

which gives after some calculation

$$\frac{\partial E}{\partial w_i} = \sum_{p=1}^{P} (O_p - d_p) x_{i,p}$$

where $x_{i,p}$ is the single input component x_i for training example p. The weight update rule for gradient descent is

$$\Delta w_i = -\eta \sum_{p=1}^{P} (O_p - d_p) x_{i,p} = \eta \sum_{p=1}^{P} (d_p - O_p) x_{i,p}$$

In the case where η is too large, the gradient descent search may overstep the minimum in the error surface rather than settling into it. One solution is to gradually reduce the value of η as the number of gradient descent steps grows. Gradient descent is a strategy for searching through a large or infinite hypothesis space which can be applied whenever

1. the hypothesis space contains continuously parameterised hypotheses.
2. the error can be differentiated with respect to these hypothesis parameters.

The key practical difficulties in applying gradient descent are

1. converging to a local minimum can sometimes be quite slow.
2. if there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.

While the gradient descent training rule computes weight updates after summing over all the training examples in $\mathcal{D}$, the stochastic gradient descent approximate this gradient descent search by updating weights incrementally, following the calculation of the error for each individual example. Hence, as we iterate through each training example, we update the weight according to

$$\Delta w_i = \eta (d - O) x_i$$

where the subscript i represents the ith element for the training example in question. One way of viewing this stochastic gradient descent is to consider a distinct error function $E_p(w)$ defined for each individual training example p as follow

$$E_p(w) = \frac{1}{2}(d_p - O_p)^2$$

where d_p and O_p are the target value and the unit output value for training example p. We therefore iterates over the training examples p in $\mathcal{D}$, at each iteration altering the weights according to the gradient with respect to $E_p(w)$. The sequence of these weight updates provides a reasonable approximation to descending the gradient with respect to the original error function $E(w)$. This training rule is known as the delta rule, or sometimes the least-mean-square (LMS) rule. Note, the delta rule converges only asymptotically toward the minimum error hypothesis, but it converges regardless of whether the training data are linearly separable.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.2 Introducing multilayer networks

While a single perceptron can only express linear decision surfaces, multilayer networks introduced in Section (4.1.2) are capable of expressing a rich variety of nonlinear decision surfaces. Since multiple layers of cascaded linear units can only produce linear functions, we need to introduce another unit to represent highly nonlinear functions. That is, we need a unit whose output is a nonlinear function of its inputs, but which is also a differentiable function of its input. We are now going to discuss in detail such a unit and then describe how to apply the gradient descent in the case of multilayer networks.

4.2.1 Describing the problem

We consider a feedforward network with N_i input units, N_o output units and N_k hidden layers which can exhibit any desired feedforward connection pattern. We are also given a training set $\{(\overline{x}_1, \overline{d}_1), ..., (\overline{x}_P, \overline{d}_P)\}$ consisting of P ordered pairs of n- and m-dimensional vectors called the input and output patterns, respectively. Note, the training set may consits of sequential data, or time-varying input, which we then represent as $\{(\overline{x}_1, \overline{d}_1), ..., (\overline{x}_T, \overline{d}_T)\}$ with T ordered pairs. We assume that the primitive functions at each node of the network is continuous and differentiable, and that the weights of the edges are real numbers selected randomly so that the output $\overline{O}_p$ of the network is initially different from the target $\overline{d}_p$. The idea being to minimise the distance between $\overline{O}_p$ and $\overline{d}_p$ for $p = 1, ..., P$ by using a learning algorithm searching in a large hypothesis space defined by all possible weight values for all the units in the network. There are different measures of error, and for simplicity we let the error function of the network be given by

$$E = \frac{1}{2} \sum_{p=1}^{P} ||\overline{O}_p - \overline{d}_p||_2^2$$

which is a sum of L_2 norms. After minimising the function with a training set, we can consider new unknown patterns and use the network to interpolate it. We use the backpropagation algorithm to find a local minimum to the error function by computing recursively the gradient of the error function and correcting the initial weights. Every one of the j output units of the network is connected to a node evaluating the function $\frac{1}{2}(O_{p,j} - d_{p,j})^2$ where $O_{p,j}$ and $d_{p,j}$ denote the jth component of the output vector $\overline{O}_p$ and the target vector $\overline{d}_p$, respectively. The outputs of the additional m nodes are collected at a node which adds them up and gives the sum E_p as its output. The same network extension is built for each pattern $\overline{d}_p$. We can then collect all the quadratic errors and output their sum, obtaining the total error for a given training set $E = \sum_{p=1}^{P} E_p$. Since E is computed exclusively through composition of the node functions, it is a continuous and differentiable function of the q weights $w_1, .., w_q$ in the network. Therefore, we can minimise E by using an iterative process of gradient descent where we need to calculate the gradient

$$\nabla E = (\frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_q})$$

and each weight is updated with the increment

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \text{ for } i = 1, .., q$$

where η is a learning constant, that is, a proportionality parameter defining the step length of each iteration in the negative gradient direction. Hence, once we have a method for computing the gradient,we can adjust the weights iteratively until we find a minimum of the error function where $\nabla E \approx 0$.

***Remark* 4.2.1** *In the case of multilayer networks, the error surface can have multiple local minima. Hence, the minimisation by steepest descent will only produce a local minimum of the error function, and not necessarily the global minimum error.*

Electronic copy available at: https://ssrn.com/abstract=3307566

4.2.2 Describing the algorithm

For simplicity of exposition we first describe the algorithm in the case where there is no hidden layer and the training set consists of a single input-output pair ($P = 1$). We assume a set of artificial neurons $\{f_{i,j}\}$, each receiving $\{x_i\}_{i=1}^{N_i}$ input and computing $\{O_j\}_{j=1}^{N_o}$ output, and we focus our attention on one of the weights, $w_{i,j}$, going from input i to neuron j in the network. We let W denote the $N_i \times N_o$ weight matrix with element $w_{i,j}$ at the ith row and the jth column and we let $w(j)$ be the $N_i \times 1$ row vector for the jth column. In the back-propagation algorithm, given the input vector $x = (x_1, .., x_{N_i})$, the activation function (also called net_j) for the jth neuron satisfies the weighted sum

$$A_j(x, w) = x \cdot w(j) = \sum_{i=1}^{N_i} x_i w_{i,j} \ , j = 1, .., N_o$$

which only depends on the inputs and the weights $w_{i,j}$ from input i to neuron j. We let the output function (or threshold box, or activation function) be a function g of the activation function, getting

$$O_j(x, w) = g(\tilde{A}_j(x, w)) \ , j = 1, .., N_o$$

where $\tilde{A}_j(x, w) = A_j(x, w) + b_j$ and b_j is a bias value. In compact form, the output vector of all units is

$$O(x, w) = g(xW)$$

using the convention that we apply the function $g(\cdot)$ to each component of the argument vector. The simplest output function is the identity function. When using a threshold activation function, if the previous output of the neuron is greater than the threshold of the neuron, the output of the neuron will be one, and zero otherwise. Further, to simplify computation, the threshold can be equated to an extra weight. The error being the difference between the actual and the desired output, it only depends on the weights. Hence, to minimise the error by adjusting the weights, we define the error function for the output of each neuron. The error function for the jth neuron satisfies

$$E_j(x, w, d) = (O_j(x, w) - d_j)^2 \ , j = 1, .., N_o$$

where d_j is the jth element the desired target vector $\overline{d}$. In that setting, the total error of the network is simply the sum of the errors of all the neurons in the output layer

$$E(x, w, d) = \frac{1}{2}||\overline{O}(x, w) - \overline{d}(x)||_2^2 = \frac{1}{2}\sum_{j=1}^{N_o} E_j(x, w, d)$$

To minimise the error, we will update the weights of the network so that the expected error in the next iteration is lower using the method of gradient descent. Each weight is updated by using the increment

$$\Delta w_{i,j} = w_{i,j}^{l+1} - w_{i,j}^{l} = -\eta \frac{\partial E}{\partial w_{i,j}^{l}}$$

where l is the iteration counter, and $\eta \in (0, 1]$ is a learning rate. The size of the adjustment depends on η and on the contribution of the weight to the error of the function. The adjustment will be largest for the weight contributing the most to the error. We repeat the process until the error is minimal.

4.2.3 Describing the nonlinear transformation

We use the chain rule to compute the gradient of the error function. Since this method requires computation of the gradient of the error function at each iteration step, we must guarantee the continuity and differentiability of the error function. One way forward is to use the sigmoid, or logistic function, a real function $f_c : \mathbb{R} \to (0, 1)$ defined as

$$f_c(x) = \frac{1}{1 + e^{-cx}}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

where the constant c can be chosen arbitrarily, and its reciprocal $\frac{1}{c}$ is called the temperature parameter. For $c \to \infty$ the sigmoid converges to a step function at the origin. Further, $\lim_{x\to\infty} f_c(x) = 1$, $\lim_{x\to-\infty} f_c(x) = 0$, and $\lim_{x\to 0} f_c(x) = \frac{1}{2}$. The first derivative of the sigmoid with respect to x is

$$\frac{d}{dx} f_c(x) = \frac{ce^{-cx}}{(1+e^{-cx})^2} = cf_c(x)(1-f_c(x)) \tag{4.2.5}$$

and the second derivative of the sigmoid with respect to x is

$$\frac{d^2}{dx^2} f_c(x) = cf_c^{'}(x) - 2cf_c(x)f_c^{'}(x)$$

where $f_c^{'}(x) = \frac{d}{dx} f_c(x)$. To get a symmetric output function, we can consider the symmetrical sigmoid $f_s(x)$ defined as

$$f_s(x) = 2f_1(x) - 1 = \frac{1-e^{-x}}{1+e^{-x}}$$

which is the *hyperbolic tangent* for the argument $\frac{x}{2}$, written $tanh(\frac{x}{2})$. An alternative solution is simply to linearly translate the domain of definition of the logistic function in the range $[b_L, b_H]$ where b_L is the lower bound and b_H the upper bound. Hence, the real function $f_{b,c} : \mathbb{R} \to (b_L, b_H)$ is defined as

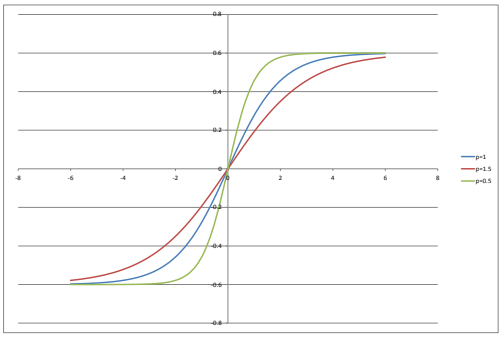
$$f_{b,c}(x) = b_L + (b_H - b_L)f_c(x)$$

with $\lim_{x\to\infty} f_c(x) = b_H$, $\lim_{x\to-\infty} f_c(x) = b_L$, and $\lim_{x\to 0} f_c(x) = \frac{1}{2}(b_L + b_H)$. The derivative of this function with respect to x is

$$\frac{d}{dx} f_{b,c}(x) = (b_H - b_L)\frac{d}{dx} f_c(x) = (b_H - b_L)cf_c(x)(1-f_c(x))$$

Hence, in order to get a symmetric output function we can set $b_L = -1.5$ and $b_H = 1.5$. Example of translated logistic function and its derivative are given in Figure (4.2) for $b_L = -0.6$ and $b_H = 0.6$ with $c = \frac{1}{p}$ for $p = 0.5, 1, 1.5$. Several other output functions can be used and have been proposed in the back-propagation algorithm. However, smoothed output function can lead to local minima in the error function which would not be there if the Heavised function had been used. Further, the slope of the sigmoid function given in Equation (4.2.5) takes values in the range $[0, \frac{1}{4}c]$ where the maximum is reached at $x = 0$. Thus, the steepest slope of the sigmoid function occurs at the origin, and depending on the size of the constant c, the slope can become greater than 1, leading to gradient explosion. In addition, the derivative of the sigmoid function is asymptotically zero for large values of x, such that the gradient will decay much more when the net value is taking large of small values. The decaying or explosion of the gradient are both called the vanishing gradient problem.

Electronic copy available at: https://ssrn.com/abstract=3307566



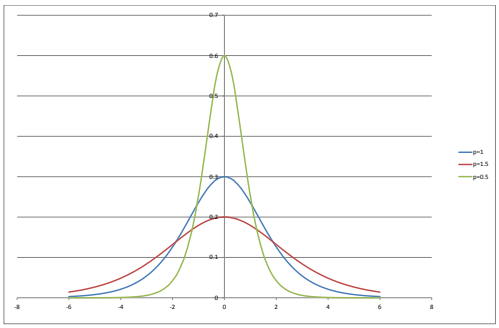


Figure 4.2: Translated logistic function and its derivative for $b_L = -0.6$ and $b_H = 0.6$.

4.2.4 A simple example

To explain the method, we first consider a two-layers ANN model, where the most common output function used with a threshold is the sigmoidal function

$$O_j(x, w) = \frac{1}{1 + e^{-A_j(x,w)}}$$

with the limits $\lim_{A_j \to \infty} = 1$, $\lim_{A_j \to -\infty} = 0$, and $\lim_{A_j \to 0} = \frac{1}{2}$ allowing for a smooth transition between the low and high output of the neuron. We compute the gradient of the total error with respect to the weight $w_{i,j}$, noted $\nabla E(w_{i,j})$, where it is understood that the subscripts i and j are fixed integers within $i = 1, ..., N_i$ and $j = 1, .., N_o$. From the linearity of the total error function, the gradient is given by

$$\frac{\partial E}{\partial w_{i,j}} = \frac{1}{2} \sum_{j=1}^{N_o} \frac{\partial E_j}{\partial w_{i,j}}, i = 1, ..., N_i$$

where the subscripts i and j are fixed integers in the partial derivatives. We first differentiate the total error with respect to the output function

$$\frac{\partial E}{\partial O_j} = (O_j - d_j)$$

and then differentiate the output function with respect to the weights

$$\frac{\partial O_j}{\partial w_{i,j}} = \frac{\partial O_j}{\partial A_j} \frac{\partial A_j}{\partial w_{i,j}} = O_j(1 - O_j)x_i$$

since $\frac{\partial A_j}{\partial w_{i,j}} = x_i$. Putting terms together, the adjustment to each weight becomes

$$\Delta w_{i,j} = -\eta(O_j - d_j)O_j(1 - O_j)x_i = \eta(d_j - O_j)O_j(1 - O_j)x_i$$

We can use the above equation to train an ANN with two layers. Given a training set with p input-output pairs, the error function can be computed by creating p similar networks and adding the outputs of all of them to obtain the total error of the set.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.3 Multi-layer back propagation

In the case of a multilayer network, again we consider a single input-output pair and assume a set of artificial neurons $\{f_{i,j,k}\}_{k=0}^{K}$ where the multilayer subscript $k = 0$ corresponds to the set of inputs $\{x_i\}_{i=1}^{N_i}$, and the remaining subscript k corresponds to the set of outputs $\{y_i\}_{i=1}^{N_{o,k}}$, where $N_{o,k}$ is the number of output in the k-th layer. In that setting, we define the output function of the jth node for the k-th layer as

$$O_{j,k}(x,w) = g(\tilde{A}_{j,k-1}(x,w)) \, , j = 1,..,N_{o,k}$$

where $\tilde{A}_{j,k-1}(x,w) = A_{j,k-1}(x,w) + b_{j,k}$ with the bias $b_{j,k}$, and we let the corresponding activation function satisfies

$$A_{j,k-1}(x,w) = \sum_{i=1}^{N_{o,k-1}} O_{i,k-1}(x,w) w_{i,j}^{k-1} \text{ and } O_{j,0}(x,w) = A_{j,-1}(x,w) = \sum_{i=1}^{N_i} x_i$$

where $w_{i,j}^{k-1}$ is the weight going from the ith node in layer $k-1$ to the jth node in layer k. Note, since $\frac{d\tilde{A}_{j,k-1}(x,w)}{dA_{j,k-1}(x,w)} = 1$, we get $\frac{\partial O_{j,k}(x,w)}{\partial A_{j,k-1}(x,w)} = \frac{\partial O_{j,k}(x,w)}{\partial \tilde{A}_{j,k-1}(x,w)}$ where $\frac{\partial O_{j,k}(x,w)}{\partial \tilde{A}_{j,k-1}(x,w)} = \frac{\partial}{\partial A_{j,K-1}} g(\tilde{A}_{j,K-1}(x,w))$.

4.3.1 The output layer

We start with the output layer K and compute the gradient $\nabla E(w_{i,j}^{k-1})$ for the weight $w_{i,j}^{k-1}$ going from the ith node in layer $(K-1)$ to the jth node in layer K. From the definition of the total error, the gradient satisfies

$$\nabla E(w_{i,j}^{k-1}) = \frac{\partial}{\partial w_{i,j}^{k-1}} E(x,w,d) = (O_{j,K} - d_j) \frac{\partial}{\partial w_{i,j}^{k-1}} (O_{j,K} - d_j)$$

since the subsripts i and j are fixed integers. Expending the output function $O_{j,K}$, we get

$$\frac{\partial}{\partial w_{i,j}^{k-1}} E(x,w,d) = (O_{j,K} - d_j) \frac{\partial}{\partial w_{i,j}^{k-1}} g\big(A_{j,K-1}(x,w) + b_{j,K}\big)$$

Using once again the chain rule, we obtain

$$\frac{\partial}{\partial w_{i,j}^{k-1}} E(x,w,d) = (O_{j,K} - d_j) \frac{\partial}{\partial A_{j,K-1}} g(\tilde{A}_{j,K-1}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-1}} A_{j,K-1}(x,w)$$

where $\tilde{A}_{j,K-1}(x,w) = A_{j,K-1}(x,w) + b_{j,K}$. The only term that depends on $w_{i,j}^{k-1}$ in the activation function $A_{j,K-1}(x,w)$ is $O_{i,K-1}(x,w) w_{i,j}^{k-1}$, and the rest of the sum will zero out after derivation. Therefore, the gradient becomes

$$\begin{aligned} \frac{\partial}{\partial w_{i,j}^{k-1}} E(x,w,d) &= (O_{j,K} - d_j) \frac{\partial}{\partial A_{j,K-1}} g(\tilde{A}_{j,K-1}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-1}} O_{i,K-1}(x,w) w_{i,j}^{k-1} \\ &= (O_{j,K} - d_j) \frac{\partial}{\partial A_{j,K-1}} g(\tilde{A}_{j,K-1}(x,w)) O_{i,K-1}(x,w) \end{aligned}$$

and we obtain the adjustment to each weight $\Delta w_{i,j}^{k-1}$. We let $e_{j,K} = \frac{\partial}{\partial O_{j,K}} E = (O_{j,K} - d_j)$ be the pre-error signal and rewrite the partial derivative as

$$\frac{\partial}{\partial w_{i,j}^{k-1}} E(x,w,d) = e_{j,K} \frac{\partial}{\partial A_{j,K-1}} g(\tilde{A}_{j,K-1}(x,w)) O_{i,K-1}(x,w)$$

Further, setting the error signal as $\delta_{j,K} = \frac{\partial}{\partial net_{j,K}} E$ where $net_{j,K} = \tilde{A}_{j,K-1}(x,w)$, we get

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\delta_{j,K} = e_{j,K} \frac{\partial}{\partial A_{j,K-1}} g(\tilde{A}_{j,K-1}(x,w))$$

and the gradient simplifies to

$$\nabla E(w_{i,j}^{k-1}) = \delta_{j,K} O_{i,K-1}(x,w)$$

so that the stochastic gradient descent rule for output units bebomes

$$\Delta w_{i,j}^{k-1} = -\eta \frac{\partial}{\partial w_{i,j}^{k-1}} E = -\eta \delta_{j,K} O_{i,K-1}(x,w)$$

We observe that the weight update rule $\Delta w_{i,j}^{k-1}$ is a multiplication of the error introduced to the output times the gradient of the output function of the current neurons input times this neurons input.

4.3.2 The first hidden layer

The next step is to consider the hidden layer $(K-1)$ and compute the gradient for the weight $w_{i,j}^{k-2}$ going from the ith node on layer $(K-2)$ to the jth node in layer $(K-1)$. Note, since the subscripts i and j are taken, for notation purpose, we use the subscript s to represent the nodes on the K-th layer. The gradient $\nabla E(w_{i,j}^{k-2})$ for the weight $w_{i,j}^{k-2}$ becomes

$$\nabla E(w_{i,j}^{k-2}) = \frac{\partial}{\partial w_{i,j}^{k-2}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial w_{i,j}^{k-2}} (O_{s,K} - d_s)$$

which gives

$$\frac{\partial}{\partial w_{i,j}^{k-2}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial w_{i,j}^{k-2}} g\big(A_{s,K-1}(x,w) + b_{s,K}\big)$$

Using once again the chain rule, we obtain

$$\frac{\partial}{\partial w_{i,j}^{k-2}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-2}} A_{s,K-1}(x,w)$$

where $A_{s,K-1}(x,w) = \sum_{j=1}^{N_{o,K-1}} O_{j,K-1}(x,w) w_{j,s}^{k-1}$. The only term that depends on $w_{i,j}^{k-2}$ in the activation function $A_{s,K-1}(x,w)$ is $O_{j,K-1}(x,w) w_{j,s}^{k-1}$, and the rest of the sum will zero out after derivation. Therefore, the gradient simplifies to

$$\frac{\partial}{\partial w_{i,j}^{k-2}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,k-1}} g(\tilde{A}_{s,K-1}(x,w)) w_{j,s}^{k-1} \frac{\partial}{\partial w_{i,j}^{k-2}} O_{j,K-1}(x,w)$$

which becomes

$$\frac{\partial}{\partial w_{i,j}^{k-2}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) w_{j,s}^{k-1} \frac{\partial}{\partial w_{i,j}^{k-2}} g(A_{j,K-2}(x,w) + b_{j,K-1})$$

Using once again the chain rule, we obtain

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\frac{\partial}{\partial w_{i,j}^{k-2}}E(x,w,d) = \sum_{s=1}^{N_{o,K}}(O_{s,K}-d_s)\frac{\partial}{\partial A_{s,K-1}}g(\tilde{A}_{s,K-1}(x,w))w_{j,s}^{k-1}\frac{\partial}{\partial A_{j,K-2}}g(\tilde{A}_{j,K-2}(x,w))\frac{\partial}{\partial w_{i,j}^{k-2}}A_{j,K-2}(x,w)$$

where $A_{j,K-2}(x,w) = \sum_{i=1}^{N_{o,K-2}} O_{i,K-2}(x,w)w_{i,j}^{k-2}$. The only term that depends on $w_{i,j}^{k-2}$ in the activation function $A_{j,K-2}(x,w)$ is $O_{i,K-2}(x,w)w_{i,j}^{k-2}$, and the rest of the sum will zero out after derivation. Therefore, given the pre-error signal $e_{s,K} = (O_{s,K} - d_s)$, we get

$$\begin{aligned}
&\frac{\partial}{\partial w_{i,j}^{k-2}}E(x,w,d) \\
=& \sum_{s=1}^{N_{o,K}} e_{s,K}\frac{\partial}{\partial A_{s,K-1}}g(\tilde{A}_{s,K-1}(x,w))w_{j,s}^{k-1}\frac{\partial}{\partial A_{j,K-2}}g(\tilde{A}_{j,K-2}(x,w))\frac{\partial}{\partial w_{i,j}^{k-2}}O_{i,K-2}(x,w)w_{i,j}^{k-2} \\
=& \sum_{s=1}^{N_{o,K}} e_{s,K}\frac{\partial}{\partial A_{s,K-1}}g(\tilde{A}_{s,K-1}(x,w))w_{j,s}^{k-1}\frac{\partial}{\partial A_{j,K-2}}g(\tilde{A}_{j,K-2}(x,w))O_{i,K-2}(x,w)
\end{aligned}$$

and we obtain the adjustment to each weight $\Delta w_{i,j}^{k-2}$. Thinking in terms of the pre-error signal, the error from the previous layer $e_{j,K-1} = \frac{\partial}{\partial O_{j,K-1}}E$ is given by

$$e_{j,K-1} = \sum_{s=1}^{N_{o,K}} e_{s,K}\frac{\partial}{\partial A_{s,K-1}}g(\tilde{A}_{s,K-1}(x,w))w_{j,s}^{k-1} = \sum_{s=1}^{N_{o,K}} \delta_{s,K}w_{j,s}^{k-1}$$

***Remark* 4.3.1** *Some authors refer to the term Downstream of unit j to describe all units whose direct inputs include the output of unit j.*

Hence, we see that the error from the previous layer is scaled down in proportion to the amount of how the previous layer influenced it. Again, the weight update rule $\Delta w_{i,j}^{k-2}$ is a multiplication of the error introduced to the output times the gradient of the output function of the current neurons input times this neurons input. That is,

$$\frac{\partial}{\partial w_{i,j}^{k-2}}E(x,w,d) = e_{j,K-1}\frac{\partial}{\partial A_{j,K-2}}g(\tilde{A}_{j,K-2}(x,w))O_{i,K-2}(x,w)$$

Setting the error signal $\delta_{j,K-1} = \frac{\partial}{\partial net_{j,K-1}}E$ where $net_{j,K-1} = \tilde{A}_{j,K-2}(x,w)$, as

$$\delta_{j,K-1} = e_{j,K-1}\frac{\partial}{\partial A_{j,K-2}}g(\tilde{A}_{j,K-2}(x,w))$$

the gradient simplifies to

$$\nabla(w_{i,j}^{k-2}) = \delta_{j,K-1}O_{i,K-2}(x,w)$$

so that the stochastic gradient descent rule for output units becomes

$$\Delta w_{i,j}^{k-2} = -\eta\frac{\partial}{\partial w_{i,j}^{k-2}}E = -\eta\delta_{j,K-1}O_{i,K-2}(x,w)$$

By iteration, we repeat this procedure until we reach $\Delta w_{i,j}^{1}$. Note, we differentiate the error function with respect to the bias value $b_{j,K}$ in the same way.

Electronic copy available at: https://ssrn.com/abstract=3307566

4.3.3 The next hidden layer

Going one step backward, we consider the hidden layer $(K-2)$ and compute the gradient for the weight $w_{i,j}^{k-3}$ going from the ith node on layer $(K-3)$ to the jth node in layer $(K-2)$. We will also assume that it corresponds to the input layer. Note, since the subscripts i and j are taken, for notation purpose, we use the subscript s to represent the nodes on the K-th layer and t to represent the nodes on the $(K-1)$-th layer. We get

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial w_{i,j}^{k-3}} (O_{s,K} - d_s)$$

which gives

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial w_{i,j}^{k-3}} g\big(A_{s,K-1}(x,w) + b_{s,K}\big)$$

Using once again the chain rule, we obtain

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} A_{s,K-1}(x,w)$$

where $A_{s,K-1}(x,w) = \sum_{t=1}^{N_{o,K-1}} O_{t,K-1}(x,w) w_{t,s}^{k-1}$. Replacing in the equation we get

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) \sum_{t=1}^{N_{o,K-1}} \frac{\partial}{\partial w_{i,j}^{k-3}} O_{t,K-1}(x,w) w_{t,s}^{k-1}$$

which gives

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) \sum_{t=1}^{N_{o,K-1}} \frac{\partial}{\partial w_{i,j}^{k-3}} g\big(A_{t,K-2}(x,w) + b_{t,K-1}\big) w_{t,s}^{k-1}$$

Using the chain rule, we obtain

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) =$$
$$\sum_{s=1}^{N_{o,K}} (O_{s,K} - d_s) \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) \sum_{t=1}^{N_{o,K-1}} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} A_{t,K-2}(x,w) w_{t,s}^{k-1}$$

where $A_{t,K-2}(x,w) = \sum_{j=1}^{N_{o,K-2}} O_{j,K-2}(x,w) w_{j,t}^{k-2}$. The only term that depends on $w_{i,j}^{k-3}$ in the activation function $A_{t,K-2}(x,w)$ is $O_{j,K-2}(x,w) w_{j,t}^{k-2}$, and the rest of the sum will zero out after derivation. Therefore, given the pre-error signal $e_{s,K} = (O_{s,K} - d_s)$, we get

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} e_{s,K} \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) \sum_{t=1}^{N_{o,K-1}} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} O_{j,K-2}(x,w) w_{j,t}^{k-2} w_{t,s}^{k-1}$$

which becomes

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) =$$
$$\sum_{s=1}^{N_{o,K}} e_{s,K} \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) \sum_{t=1}^{N_{o,K-1}} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} g(A_{j,K-3}(x,w) + b_{j,K-2}) w_{j,t}^{k-2} w_{t,s}^{k-1}$$

Setting

$$\hat{e}_{t,K-1} = e_{s,K} \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) w_{t,s}^{k-1}$$

we get

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{s=1}^{N_{o,K}} \sum_{t=1}^{N_{o,K-1}} \hat{e}_{t,K-1} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} g(A_{j,K-3}(x,w) + b_{j,K-2}) w_{j,t}^{k-2}$$

Note, from linearity we can interchange the summation operators and rewrite the above equation as

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{t=1}^{N_{o,K-1}} \sum_{s=1}^{N_{o,K}} \hat{e}_{t,K-1} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} g(A_{j,K-3}(x,w) + b_{j,K-2}) w_{j,t}^{k-2}$$

We then recover the pre-error term $e_{t,K-1} = \frac{\partial}{\partial O_{t,K-1}} E$ as

$$e_{t,K-1} = \sum_{s=1}^{N_{o,K}} \hat{e}_{s,K} = \sum_{s=1}^{N_{o,K}} e_{s,K} \frac{\partial}{\partial A_{s,K-1}} g(\tilde{A}_{s,K-1}(x,w)) w_{t,s}^{k-1}$$

such that the equation simplifies to

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{t=1}^{N_{o,K-1}} e_{t,K-1} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} g(A_{j,K-3}(x,w) + b_{j,K-2}) w_{j,t}^{k-2}$$

Using once again the chain rule, we obtain

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = \sum_{t=1}^{N_{o,K-1}} e_{t,K-1} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial A_{j,K-3}} g(\tilde{A}_{j,K-3}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} A_{j,K-3}(x,w) w_{j,t}^{k-2}$$

where $A_{j,K-3}(x,w) = \sum_{i=1}^{N_{o,K-3}} O_{i,K-3}(x,w) w_{i,j}^{k-3}$. The only term that depends on $w_{i,j}^{k-3}$ in the activation function $A_{j,K-3}(x,w)$ is $O_{i,K-3}(x,w) w_{i,j}^{k-3}$, and the rest of the sum will zero out after derivation. Therefore, the gradient simplifies to

$$\begin{aligned}
&\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) \\
= &\sum_{t=1}^{N_{o,K-1}} e_{t,K-1} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial A_{j,K-3}} g(\tilde{A}_{j,K-3}(x,w)) \frac{\partial}{\partial w_{i,j}^{k-3}} O_{i,K-3}(x,w) w_{i,j}^{k-3} w_{j,t}^{k-2} \\
= &\sum_{t=1}^{N_{o,K-1}} e_{t,K-1} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) \frac{\partial}{\partial A_{j,K-3}} g(\tilde{A}_{j,K-3}(x,w)) O_{i,K-3}(x,w) w_{j,t}^{k-2}
\end{aligned}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

Writing the error from the previous layer in terms of the Downstream of unit j as

$$e_{j,K-2} = \sum_{t=1}^{N_{o,K-1}} e_{t,K-1} \frac{\partial}{\partial A_{t,K-2}} g(\tilde{A}_{t,K-2}(x,w)) w_{j,t}^{k-2}$$

where $e_{j,K-2} = \frac{\partial}{\partial O_{j,K-2}} E$. Then, the gradient becomes

$$\frac{\partial}{\partial w_{i,j}^{k-3}} E(x,w,d) = e_{j,K-2} \frac{\partial}{\partial A_{j,K-3}} g(\tilde{A}_{j,K-3}(x,w)) O_{i,K-3}(x,w)$$

which corresponds to the multiplication of the error introduced to the output times the gradient of the output function of the current neurons input times this neurons input. Further, setting the error signal $\delta_{j,K-2} = \frac{\partial}{\partial net_{j,K-2}} E$ where $net_{j,K-2} = \tilde{A}_{j,K-3}(x,w)$, as

$$\delta_{j,K-2} = e_{j,K-2} \frac{\partial}{\partial A_{j,K-3}} g(\tilde{A}_{j,K-3}(x,w))$$

the gradient simplifies to

$$\nabla(w_{i,j}^{k-3}) = \delta_{j,K-2} O_{i,K-3}(x,w)$$

and the stochastic gradient descent rule for the next hidden layer units bebomes

$$\Delta w_{i,j}^{k-3} = -\eta \frac{\partial}{\partial w_{i,j}^{k-3}} E = -\eta \delta_{j,K-2} O_{i,K-3}(x,w)$$

This is the general rule for updating internal unit weights in arbitrary multilayer networks. Hence, the error signal travels from the output layer to the input layer. Further, the weights influence the error by some degree, and they must be taken into consideration when propagating the error.

4.3.4 Some remarks

Since the error surface for multipayer networks may contain many different local minima, the backpropagation algorithm can only converge toward some local minima in E which is not necessarily the global minimum error. Nonetheless, when the gradient descent falls into a local minimum with respect to one of these weights, it will not necessarily be in a local minimum with respect to other weights. Hence, higher dimensions might provide escape routes to the steepest descent to continue searching the space of possible network weights. In addition, the sigmoid threshold function being approximately linear when the weights are close to zero, we can initialise the network weights to values near zero so that in the early steps the network will represent a very smooth function approximately linear in its inputs. There exists several heuristic to avoid being stuck in a local minima such as adding a momentum term to the weight-update rule or using a stochastic gradient descent. One of the best approach is to train multiple networks using the same data, but initialising each network with different random weights. One can then select the best network according to one of these two methods

1. select the network with the best performance over a separate validation data set.
2. all networks can be retained and treated as a committee of networks whose output is the weighted average of the individual network outputs.

Various authors investigated the backpropagation algorithm to find out which function classes could be described by which types of networks. Three general results are known

Electronic copy available at: https://ssrn.com/abstract=3307566

- Boolean functions: every boolean function can be represented exactly by some network with two layers of units, although the number of hidden units required grows exponentially in the worst case with the number of network inputs.
- Continuous functions: every bounded continuous function can be approximated with arbitrarily small error (under a finite norm) by a network with two layers of units (see Cybenko [1989]).
- Arbitrary functions: any function can br approximated to arbitrary accuracy by a network with three layers of units (see Cybenko [1988]). This is because any function can be approximated by a linear combination of many localised functions having value 0 everywhere except for some small region, and that two layers of sigmoid units are sufficient to produce good local approximations.

The hypothesis space for backpropagation is the n-dimensional Euclidean space of the n network weights. Further, as opposed to decision tree where the hypothesis space is discrete , it is continuous for backpropagation leading to a well-defined error gradient. Also, the inductive bias of backpropagation learning is characterised by smooth interpolation between data points. An important property of backpropagation is its ability to discover useful intermediate representations at the hidden unit layers inside the network. This ability of multilayer networks to automatically discover useful representations at the hidden layers is a key feature of ANN learning as it provides extra degree of flexibility to invent features not explicitly introduced by the human designer.

4.4 Summarising the feedforward ANN

As observed in Section (4.3), when running backpropagation, the error signal travels from the output layer to the input layer. Given a hidden layer with subscript k, and focusing on neuron with index h, we define the pre-error signal and error signal as

$$\begin{aligned} e_{h,k} &= \frac{\partial}{\partial O_{h,k}} E \\ \delta_{h,k} &= \frac{\partial}{\partial \tilde{A}_{h,k-1}(x,w)} E = \frac{\partial}{\partial net_{h,k}} E \end{aligned}$$

Using these results, we are going to summarise the feedforward neural network by considering the multilayer perceptron (MLP).

4.4.1 Forward pass

We consider an MLP consisting of K hidden layers with N_i input units, $N_{o,k}$ output units in the k-th hidden layer and $N_{o,K}$ output units in the output layer. In each unit of the hidden layer, or in the output layer, we first calculate the weighted sum of the incoming values, which is referred to as the *net value* of the input unit. The *activation function*, denoted by f, is then applied to the net value. The value of the activation function, y, is the output of the unit. The activation function is required to be bounded, differentiable and monotonous. The two most common functions used in machine learning are the *hyperbolic tangent* and the *logistic sigmoid* functions, which are both nonlinear. This important feature makes it possible for the network to model nonlinear equations. As the number of hidden layers increases, the network can approximate more complex nonlinear functions. Methods using a network with a large number of hidden layers are referred to as *deep learning* network. Since the input domain of the activation function is infinite, while the output domain is finite, the activation functions are also termed as *squashing functions*. Letting h be the index of the first hidden unit, the net value and output function satisfy

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} net_{h,1} &= \sum_{i=1}^{N_i} w_{ih} x_i \\ y_h &= f(net_{h,1}) \end{aligned} \tag{4.4.6}$$

Considering two adjacent layers, denoted by $k-1$ and k, with index h and h', respectively, the summation and the activation process are similar to that of Equation (4.4.6), given by

$$\begin{aligned} net_{h',k} &= \sum_{h=1}^{N_{o,k-1}} w_{hh'} y_h \\ y_{h'} &= f(net_{h',k}) \end{aligned} \tag{4.4.7}$$

The net value and activation function of the output layer are calculated in the same way as those in the hidden layer. We let s be the index ranging over the output layer, and $N_{o,K-1}$ is the number of neurons in the $(K-1)$-th hidden layer closest to the output layer. We get

$$\begin{aligned} net_{s,K} &= \sum_{h=1}^{N_{o,K-1}} w_{hs} y_h \\ y_s &= f(net_{s,K}) \end{aligned} \tag{4.4.8}$$

4.4.2 Backward pass

In general, after a forward pass the network output is not the target input, and we need to measure the distance between the actual output and the target output to serve as a measurement of the network performance. This distance is called the *error function*. Given the input-target pair (x, z), and the network output y, then the error function is given by

$$E(x,z) = \frac{1}{2} \sum_{s:output} (y_s - z_s)^2, \tag{4.4.9}$$

where $y = (y_1, \cdots, y_s, \cdots, y_K)$, $z = (z_1, \cdots, z_s, \cdots, z_K)$, and $N_{o,K}$ is the number of units in the output layer. As long as the error function is larger than a given tolerence level, we need to modify the weights of the network to further decrease the measure E. The algorithm designed for stepwise updating the weights that minimise the error function is called the *training algorithm*, or *learning algorithm* of the network. It is natural to relate the weights updates to the *gradient*, which is a vector of derivatives of the error function with respect to all the weights. It costs the network certain number of steps to reach a tolerably small error. Putting all the weights in a vector, the weight vector at step n is denoted by w^n, which is then updated by an amount Δw^n. We denote the gradient as $\nabla E = \frac{\partial E}{\partial w^n}$ and apply the *gradient descent* repeatedly using the chain rule. By working backward, from computing the derivative with respect to the output layer to computing the derivatives with respect to all the internal weights, the error is back propagated in the network. This procedure is called the standard *back propagation*, discovered independently by different researchers (see Werbos [1974], Parker [1985], Rumelhart et al. [1986b]). In standard back propagation, the weight update of the n-th step is

$$\Delta w^n = -\eta \frac{\partial E}{\partial w^n}, \tag{4.4.10}$$

where η is the *learning rate*, a real number between 0 and 1. Following the previous notations, the detailed procedure to calculate the gradient $\frac{\partial E}{\partial w_{ij}}$ is as follows:

1. Calculate the derivatives of the error function with respect to the output units:

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\frac{\partial E}{\partial y_s} = y_s - z_s \tag{4.4.11}$$

$$\frac{\partial E}{\partial net_{s,K}} = \frac{\partial E}{\partial y_s}\frac{\partial y_s}{\partial net_{s,K}} \tag{4.4.12}$$

2. Calculate the derivatives of the error function with respect to the hidden units. Work backward through hidden layers, using the chain rule. To do so, we introduce the *error signal*:

$$\delta_j := \frac{\partial E}{\partial net_j}, \tag{4.4.13}$$

where j is the index of an arbitrary unit in the network.

(a) Calculate the error signals of the last hidden layer:
Let j be the index of a unit in the last hidden layer which is the closest to the output layer. Since the error E depends on hidden unit j only through its connection to the output units, we have

$$\begin{aligned}\delta_j &= \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_{j,K-1}} \\ &= \frac{\partial y_j}{\partial net_{j,K-1}}\sum_{s=1}^{N_{o,K}}\frac{\partial E}{\partial net_{s,K}}\frac{\partial net_{s,K}}{\partial y_j}\end{aligned} \tag{4.4.14}$$

From Equation (4.4.7) and (4.4.8), and given the definition of the error signal in Equation (4.4.13), we have,

$$\delta_j = f^{'}(net_{j,K-1})\sum_{s=1}^{N_{o,K}}\delta_s w_{js} \tag{4.4.15}$$

(b) Calculate the error signals for hidden layers before the last hidden layer:
The error signals for the hidden units in hidden layer k ($k < K$) are based on the calculation of the error signal of the $(k+1)$-th layer due to the chain rule. That is to say, except for the units of the last hidden layer, the error signal of each unit in other hidden unit can be computed recursively as

$$\delta_i = f^{'}(net_{i,k})\sum_{j=1}^{N_{o,k+1}}\delta_j w_{ij} \tag{4.4.16}$$

where i and j are indicators of unit in the hidden layers k and $k+1$, respectively, before the last hidden layer.

Having computed all the error signals for all the hidden units as well as the one for the output units, the derivatives of error function with respect to the weights w_{ij} can be written as:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial net_{j,k+1}}\frac{\partial net_{j,k+1}}{\partial w_{ij}} = \delta_j y_i \tag{4.4.17}$$

Then, the change of weight from unit i to unit j is given by

$$\Delta w_{ij} = -\alpha\delta_j y_i \tag{4.4.18}$$

Thus, introducing the concept of error signal makes the back propagation process more comprehensible, as only the error signal propagates backward through the network. The changing amount of the weight is proportional to the

Electronic copy available at: https://ssrn.com/abstract=3307566

product of the error signal of its destination unit and the output of its source unit. To summarise, the forward pass calculates the network output, while the backward pass updates the weights to minimise the error. The method used in backward pass is called back propagation, and a forward pass together with a backward pass is regarded as one *loop*. The training process continue until some *stopping criteria* is met, such as the error function is small enough, or, it stops to decrease after a certain number of loops.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 5

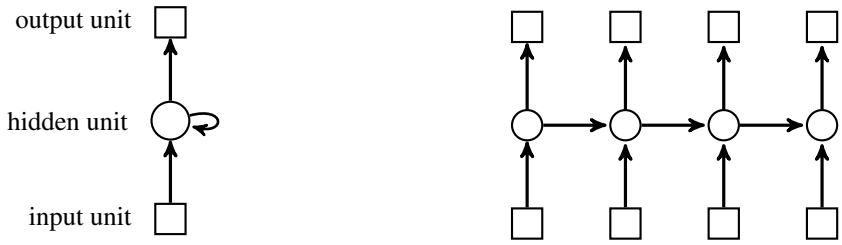
Recurrent neural networks

5.1 Presenting recurrent neural networks

5.1.1 An overview

While the feedforward neural networks (FNNs) discussed in Section (4.3) have no cycles for connections, a *Recurrent Neural Network* (RNN) has feedback connection, meanning that the nodes of the network have cyclical connections between them (see Figure 5.1a). The existence of cycles allows RNNs to develop a self-sustained temporal activation dynamics along its recurrent connection pathways, even in the absence of input, making them a dynamical system. There are two main classes of RNNs, the first one being characterised by an energy-minimising stochastic dynamics and symmetric connections (see Taylor et al. [2007]), and the second featuring a deterministic update dynamics and directed connections. We are going to concentrate on the latter. As shown in Figure 5.1b, RNNs can be visualised by unfolding the RNNs along the whole input sequence. That is, in the case of time series (notion of time), the RNN are unfolded through time (see Figure (5.2)). The unfolded graph has no cycles, which is the same as FNNs, so that the forward pass and backward pass of MLP can be applied. Recall that the MLP is an approximator for nonlinear functions, and since RNN can be unfolded to a deep feedforward network, it has the advantage of the MLP, making it a better approximator. The biggest feature of recurrent neural network is that it has a sparsely connected hidden layer called reservoir, which enables RNNs to have short term memory that captures information about what has been calculated so far. Usually, this recurrent part of the structure is essential in learning complex patterns. In fact, RNNs have proved to be an attractive form for modelling non-linearity due to their ability to approximate any dynamical system with arbitrary precision (see Siegelmann et al. [1991]). Further, Funahashi et al. [1993] showed that under mild and general assumptions, RNNs are universal approximatiors of dynamical systems. Indeed, as an equivalent theory of universal approximation for MLPs, it is said that a single hidden layer RNN with sufficient hidden units can approximate any sequence to arbitrary accuracy (see Hammer [2000]). For simplicity of exposition, we focus on a simple RNN with one self-connected hidden layer, as shown in Figure 5.1a. As trivial the difference in topology between FNNs and RNNs may seems, the advantage of the RNNs over the FNNs is profound. The latter can only map a limited number of inputs to a limited number of outputs, while a simple RNN can map the entire time series to some outputs. This is significant for time series prediction, as the long history can be fed to the network, and the existence of the recursive connection allows the network to memorise information of previous time steps, which is useful considering that financial time series are mostly serially correlated.

Electronic copy available at: https://ssrn.com/abstract=3307566



(a) A simple RNN (b) RNN unfolded

Figure 5.1: Recurrent Neural Network

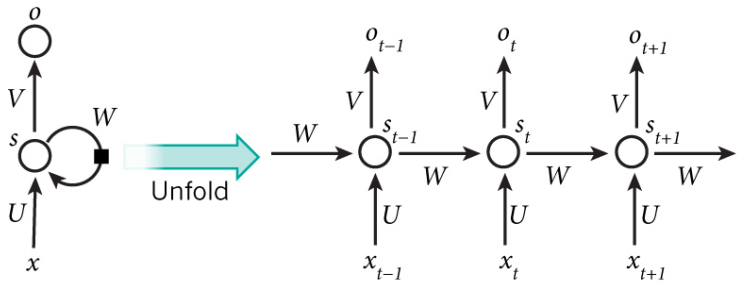


Figure 5.2: Recurrent Neural Network in time

5.1.2 The algorithm

5.1.2.1 Forward pass

We now consider a sequence of length T with N_i inputs, one hidden layer ($K = 2$) with $N_{o,K-1}$ hidden units and N_K output units. The i-th external input at time t is denoted by $x_i(t)$, and we let $net_{h,K-1}(t)$ and $y_h(t)$ be the *net value* (summation of the value received) and the output of unit h, respectively, in the hidden layer at time t. After unfolding the network, the forward pass of the RNN is similar to that of a MLP. The only difference being that the unit in the hidden layer receives values from both the current external inputs and the previous output of all hidden units. Mathematically, the net value of a hidden unit is given by

$$net_{h,K-1}(t) = \sum_{i=1}^{N_i} w_{ih} x_i(t) + \sum_{h'=1}^{N_{o,K-1}} w_{h'h} y_{h'}(t-1) \tag{5.1.1}$$

where $h^{'}$ is a unit on the previous hidden layer. Then the output value of the hidden unit is calculated by applying the sigmoid activation f to the net value

$$y_h(t) = f(net_{h,K-1}(t)) \tag{5.1.2}$$

such that the output of hidden units at each time step can be computed recursively, starting from the first time step $t = 1$. Initial value of $y_h(0), \forall h$ is commonly set to be zero, meaning that the network has not received information before the beginning of forward pass. Still, some researchers tend to have nonzero intial value by which they found that the network is more stable (see Zimmermann [2006]). The net value of the output unit, $net_{s,K}$, depends only on the output value of the hidden layer so that the output units are synchronised with the hidden units

Electronic copy available at: https://ssrn.com/abstract=3307566

$$net_{s,K}(t) = \sum_{h=1}^{N_{o,K-1}} w_{hs} y_h(t)$$

The error function can be defined as

$$E = \sum_{t=1}^{T} \sum_{s=1}^{N_K} \frac{1}{2}(y_s(t) - z_s(t))^2 \tag{5.1.3}$$

where $z_s(t)$ is the target value of the output unit s at time t. However, this is not the unique form of error function, as it should be based on the training algorithms of the RNN, which we will discuss in the next section.

5.1.2.2 Backward pass

We explained in Section (4.3) how to calculate the derivatives of the error function with respect to the weights in an MLP. In the case of RNNs, there are mainly two algorithms to calculate the weight derivatives: the Real Time Recurrent Learning (RTRL) (see Robinson et al. [1987], Williams et al. [1989]) and the Backpropagation Through Time (BPTT) (see Werbos [1990], Williams et al [1992]). In ordrer to describe the learning algorithm of the RNN and detail its associated problems, we focus on the BPTT, as it is simple conceptually and more efficient in computation time (see Graves [2005]). Figure 5.3 illustrates the scheme for updating the weights. Note that the same weights are used in every time step. The error function is defined in Equation (5.1.3), which include all the time steps.

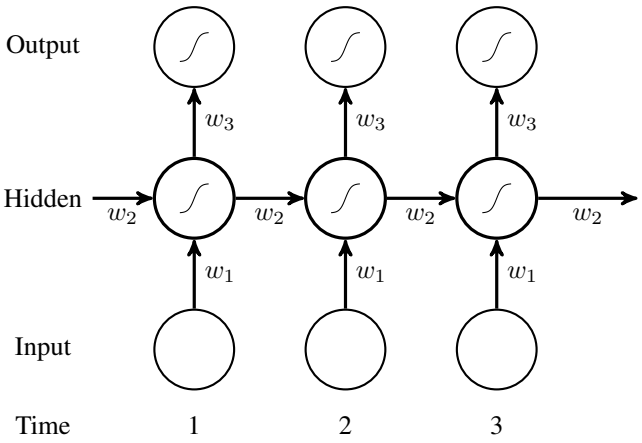


Figure 5.3: Weights in BPTT

For MLPs, the error signals are computed in Equations (4.4.12), (4.4.14) and (4.4.16) according to the type of unit. However, what is common for the error signals of the hidden units is that they depend on the sum of error signals that flow back to them. Since unfolded RNNs are equivalent to MLPs, such rule can also be applied. As illustrated in Figure 5.3, we consider the hidden unit h_2 at time $t = 2$, which has two outgoing connections. We can see that the error signal of hidden unit h_3 at time $t = 3$, and that of the output unit o_2 at time $t = 2$ would flow back to unit h_2 during back propagation through time. The red dashed arrows in Figure 5.4 illustrates the flow of error signals backward to a particular hidden unit. Hence, the error signal at unit h_2 can be computed as

$$\delta_{h_2} = \delta_{h_3} + \delta_{o_2}$$

Generally, the error function depends on the output of hidden layer not only through the current output layer, but also through the hidden layer next time. Thus, we have

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\delta_h(t) = f^{'}(net_{h,K-1}(t))\Big(\sum_{s=1}^{N_K}\delta_s(t)w_{hs} + \sum_{h'=1}^{N_o}\delta_{h'}(t+1)w_{hh'}\Big) \tag{5.1.4}$$

where $\delta_j(T+1) = 0$, $\forall j$. Thus, starting the backward pass at the last time T, we apply Equation (5.1.4) recursively. Having computed the error signals, we can now calculate the derivative of the error function with respect to the weight via chain rule, getting

$$\frac{\partial E}{\partial w_{ij}} = \sum_{t=1}^{T}\frac{\partial E}{\partial net_j(t)}\frac{\partial net_j(t)}{\partial w_{ij}} = \sum_{t=1}^{T}\delta_j(t)y_i(t) \tag{5.1.5}$$

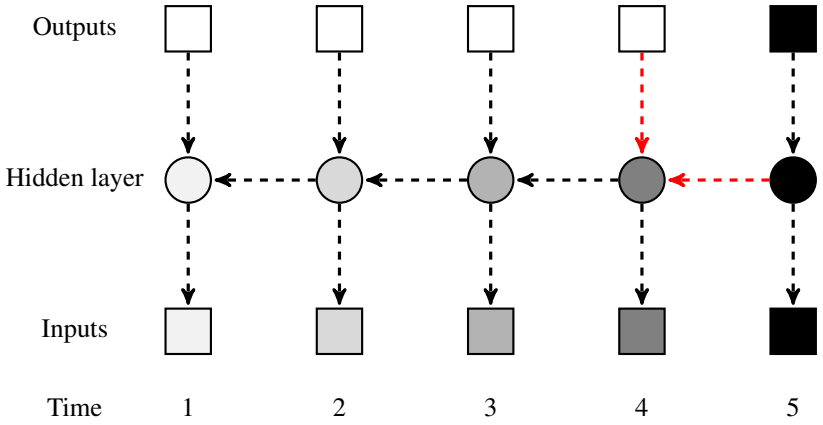


Figure 5.4: The flow of error signals and vanishing gradient problem in back propagation. The darkness of the shading of the nodes in the unfolded recurrent network indicates the degree of influence (sensitivity) of error signal of output unit at time $t = 5$. The darker the shade represents higher sensitivity. The sensitivity decays as the error signal is back propagated through time.

Electronic copy available at: https://ssrn.com/abstract=3307566

5.2 The long short-term memory

The impact of RNNs in nonlinear modelling has been limited because they are difficult to train by gradient-descent methods. The gradual change of network parameters during the learning process leads the network to bifurcations, where the gradient information degenerates and may become ill-defined (see Doya [1992]). Further, many update cycles may be necessary to obtain convergence of a few parameters, leading to long training times. In addition, when dealing with long-range memory, the gradient information may dissolves exponentially over time. One remedy is to use the Long Short-Term Memory networks (LSTM), which we are now going to describe.

5.2.1 The vanishing gradient problem

5.2.1.1 Description

Even though recurrent neural networks (RNNs) are capable of processing serially correlated sequences, the length of sequence that a standard recurrent neural network can access is actually very limited. It results from the fact that the gradients would either decay or blow up when cycling around the recursive connections for too many times, which is usually referred to as *vanishing gradient problem* (see Bengio et al. [1994]). We can see in Equation (5.1.5) that the derivative of the error with respect to a weight depends on the error signal of the weight's destination unit, j, which itself depends on the derivative of the activation function. Recall, the derivative of the sigmoid function can explode or vanish depending on the slope of the function. As the number of time steps get larger and larger, so does the number of layers in the unfolded RNN. Assuming $c = 1$ in the sigmoid function, the error signal of a hidden unit at time $t = T$ would be propagated back through $(T - 1)$ layers, until a hidden layer at time $t = 1$. That is, it would be multiplied by $(T - 1)$ sigmoid activations' derivatives, all in the range $[0, \frac{1}{4}]$. For T not too large, the vanishing problem does not have much impacts on the network, but for larger T the gradient would decay exponentially. Similarly, for large enough c, the derivative of the sigmoid function near the origin ($x = 0$) would be larger than 1, leading to gradient explosion. This property of sigmoid function is illustrated in Figure 5.5. For these reasons, training RNNs with standard gradient descent algorithm is only feasible for small time steps. For longer time dependencies, the gradient vanishes as the error signal is propagated back through time, so that the network weights are never adjusted correctly when taking the events far back in the past into account (see Hochreiter et al. [2001]).

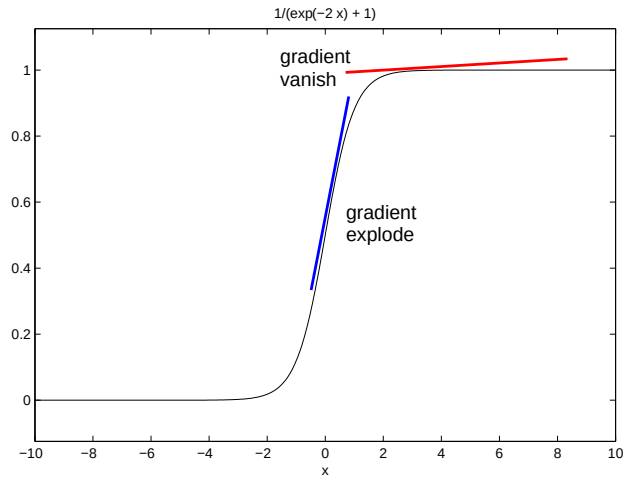


Figure 5.5: Plot of $\frac{1}{1+e^{-2x}}$ and its tangent lines. The black curve is the sigmoid function ranged $[0, 1]$ with $c = 2$. The red line is the flat tangent line.

Electronic copy available at: https://ssrn.com/abstract=3307566

5.2.1.2 The constant error carousel

One consequence of the vanishing, or exploding, gradient problem described above is that the traditional RNN will fail when involved with large time series. Since the vanishing gradient problem stems from the fact that the tangent line of the sigmoid function is either flat or steep, it is reasonable for some units to have constant gradient equating to 1. This kind of unit has been presented as the *Constant Error Carousel* (CEC) (see Hochreiter et al. [1997]). To describe the advantageous properties of CEC, we consider a single unit, j, with a single connection to itself. According to the rule of calculation of error signals, the jth error signal satisfies

$$\delta_j(t) = f_j^{'}(net_j(t))\delta_j(t+1)w_{jj}$$

where $f_j(\cdot)$ is the activation function of unit j. A constant error flow implies that $\delta_j(t) = \delta_j(t+1)$, leading to

$$f_j^{'}(net_j(t))w_{jj} = 1$$

This is an ordinary differential equation (ODE)

$$\frac{\partial f_j(net_j(t))}{\partial net_j(t)} = \frac{1}{w_{jj}}$$

which we can integrate, getting

$$f_j(net_j(t)) = \frac{net_j(t)}{w_{jj}}$$

for arbitrary $net_j(t)$. Thus, the activation function f_j for the jth unit should be **linear**

$$y_j(t+1) = f_j(net_j(t+1))$$

Since unit j has only one connection to itself, we get

$$net_j(t+1) = w_{jj}f_j(t)$$

and from the equation of $f_j(net_j(t))$, we get

$$f_j(w_{jj}y_j(t)) = \frac{w_{jj}y_j(t)}{w_{jj}} = y_j(t)$$

We can therefore set the activation function to be the identity function $f_j(x) = x, \forall x$, obtaining $w_{jj} = 1$. The unit j above can be extended to CEC by adding some extra features. A multiplicative *input gate unit* and a multiplicative *output gate unit* are introduced to control the input and output of the CEC, respectively. The input gate unit can prevent the memory of unit j from being overwritten by irrelevant inputs, while the output gate unit avoid perturbation of other units by controlling the output of unit j. The CEC ensures that the error signal arriving at the memory cell would not be scaled up or down during back propagation, and thus can avoid exponential gradient.

5.2.2 Network architecture

A unit which include the input gate unit, the output gate unit and the CEC is called a *memory cell* (see Figure 5.6). We are now going to describe the *Long Short-Term Memory* (see Hochreiter et al. [1997]) which is a recurrent neural network consisting of set of *memory blocks* where each block contains one or more memory cells. In fact, the LSTM network is the same as that of a standard recurrent neural network, except that the conventional hidden units are replaced by a memory block with cells and gates encapsulated in. Surely, the hidden layer of the LSTM network can be a mixture of conventional hidden units and memory blocks, but the former is not necessary. The v-th memory cell of the j-th memory block is denoted by c_j^v, the net value of the cell, the input gate and the output gate are $net_{c_j^v}$,

Electronic copy available at: https://ssrn.com/abstract=3307566

net_{in_j} and net_{out_j}, respectively, the output value of the cell, the input gate and the output gate are $y^{c_j^v}$, y^{in_j} and y^{out_j}, respectively. Note that memory cells in the same memory block are controlled by the same input gate and output gate (see Figure 5.6).

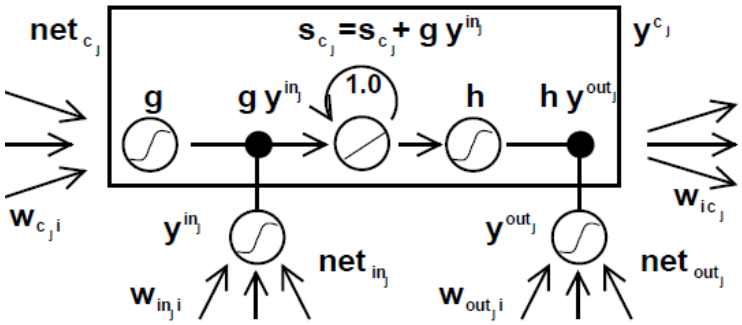


Figure 5.6: Architecture of a memory block with one memory cell. The CEC is a central linear unit with a self-connected weight 1.0. gate units use inputs from other units to decide whether to access or discard certain information.

As with a standard recurrent network, we have

$$y^{in_j}(t) = f_{in_j}(net_{in_j}(t))$$

$$y^{out_j} = f_{out_j}(net_{out_j}(t))$$

where f_{in_j} and f_{out_j} are the activation function of the input gate and output gate of memory block i, and

$$\begin{aligned} net_{in_j}(t) &= \sum_u w_{in_j u} y^u(t-1) + \sum_{n=1}^{N_i} w_{in_j n} x_n(t) \\ net_{out_j}(t) &= \sum_u w_{out_j u} y^u(t-1) + \sum_{n=1}^{N_i} w_{out_j n} x_n(t) \\ net_{c_j^v}(t) &= \sum_u w_{c_j^v u} y^u(t-1) + \sum_{n=1}^{N_i} w_{c_j^v n} x_n(t) \end{aligned}$$

where the superscript u stands for memory blocks and other traditional hidden unit. The output of the memory cell is different from that of a conventional hidden layer. An additional variable $s_{c_j^v}$, called *internal state*, should be considered

$$\begin{aligned} s_{c_j^v}(t) &= s_{c_j^v}(t-1) + y^{in_j}(t) g(net_{c_j^v}(t)),\ t > 0 \\ s_{c_j^v}(0) &= 0, \end{aligned} \tag{5.2.6}$$

Actually, the first line of Equation (5.2.6) can be written as

$$s_{c_j^v}(t) = 1.0 \times s_{c_j^v}(t-1) + y^{in_j}(t) g(net_{c_j^v}(t))$$

where 1.0 corresponds to the self-recursive connection with weight 1.0 in Figure 5.6, which is essentially w_{jj} obtained by the introduction of a simple self-connected unit j without the vanishing gradient problem. So the output of the cell can be computed as

Electronic copy available at: https://ssrn.com/abstract=3307566

$$y^{c_j^v}(t) = y^{out_j}(t)h(s_{c_j^v}(t))$$

where g and h are the squashing function of the net value of memory cell and the internal sate $s_{c_j^v}$.

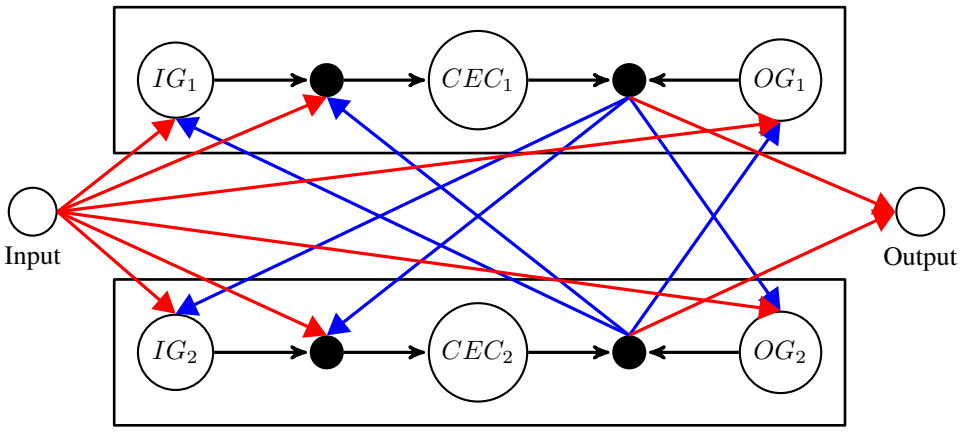


Figure 5.7: Connections in LSTM network. The example network consists of one input unit, a hidden layer of two single-cell LSTM memory blocks and one output units. Suppose the input is at time t. The the sources of the blue connections are from the previous one time step $t-1$. while the red connections convey information of current time step t. IG:input gate, OG: output gate, CEC:constant error carrousel

It is important that the gate units and the internal states are only visible within the cell, and that only the output of the cell can connect to other blocks (including gates and cell) in the hidden layer. Therefore, the net value is the summation of current input values and the output of memory cells $y^{c_i^j}$ from previous time step. Further, Graves [2005] made some modifications to the original model of Schmidhuber and Hochreiter by modifying the rule connecting the units. Only the outputs memory cells are allowed to connect to other blocks, and the CECs and outputs of gate units are only visible within the memory blocks they are located in. As an example, Figure 5.7 shows the types of connections between two blocks. The gates units enable the memory cells in LSTM to store as well as to access information for a long period of time, and thus alleviate the vanishing gradient problem. For instance, if the activation function of the input gate is zero (the input gate is closed), then the memory of the cell will not be overwritten by the current input, thereby making it accessible to the network at a later time, as long as the output gate is open. However, the original LSTM has a weakness. When the sequence of time is long and have not been segmented to reset the network at certain time, Equation (5.2.6) implies that the internal state will grow infinitely and the network will collapse. To remedy this problem, Gers et al. [2000] proposed to add a *forget gate* to the self-connection with weight 1.0 in the original LSTM cell (see Figure 5.6). Then the revised equation for the internal state is extended from Equation (5.2.6) as follow

$$\begin{aligned} s_{c_j^v}(t) &= y^{\phi_j}(t)s_{c_j^v}(t-1) + y^{in_j}(t)g(net_{c_j^v}(t)),\ t > 0 \\ s_{c_j^v}(0) &= 0. \end{aligned} \tag{5.2.7}$$

where $y^{\phi_i}(t)$ is the value of the forget gate of the i-th memory block, which is squashed between zero and one. The forget gate is analogous to the reset operation of the memory cell. So far, the gate units are not entitled to control the internal state, as the gates have merely two sources of input: from the current input units and the previous output of all memory cells. In other words, the gates unit can only observe the cells' output. Once the output gate is closed, there is no way for gates to access the CEC they are supposed to control, which results in insufficient information that do harm to the performance of the network. In order to avoid the lack of information of internal states, another augmentation of the LSTM with forget gate was also introduced by Gers et al. [2002]. Weighted *peephole weights* are added in order to connect from the CEC to all gates of the same memory block. This effective remedy ensures that the all gates can "inspect" the internal state currently, even if the output gate is closed (see the dashed arrow in Figure 5.8).

Electronic copy available at: https://ssrn.com/abstract=3307566

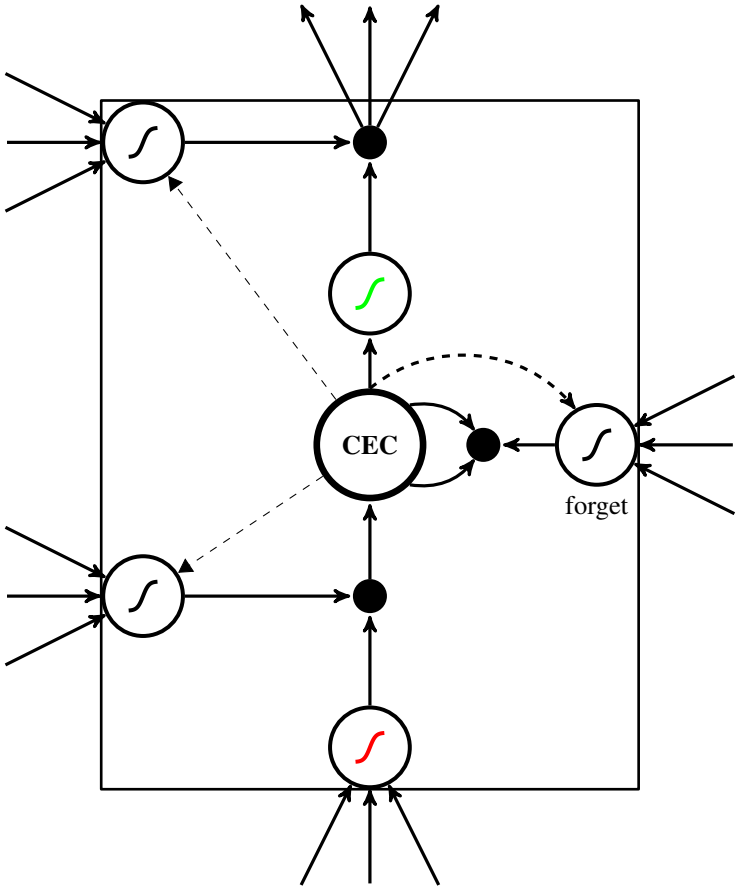


Figure 5.8: A modern LSTM memory block with one cell. The input, output and forget gates are the analogous of write read and reset operations for the cells. The three gates are nonlinear summation units that collect outputs from outside and inside the block. The small black dots are multiplicative units by which the activation of the cell is controlled. The gate units have sigmoid function f with range $[0, 1]$, so that 0 corresponds to the gate being closed and 1 to the gate being open. The unit with red S-shaped curve is the activation function that squashes the net value of cells' input, and the unit with green S-shaped curve is the activation function of cells' output. Usually, the functions g and h are task-specific and are encouraged to have ranges different from $[0, 1]$. The input and output gates multiply the input and output of the cell, and the forget gate multiply the previous internal state. Peephole weights from internal states to gates are shown by dashed arrow. Note, except for the self-recursive connection of the CEC, connections within the blocks have fixed weights of 1.0. The only outputs from the memory block to the rest of the network is from the upper multiplicative unit.

We can summarise the types of connections of the modern LSTM as follow

- Outgoing connections
  - outside memory block
    Cells' output can feed to any types of units in any blocks in the hidden layer and the output units and they are the only output that can be connected to other blocks and output units.

Electronic copy available at: https://ssrn.com/abstract=3307566

  - inside memory block
    The gate unit can only pass value forward to cells of the block where the gate belong to. The internal state can connect to all types of gate within the same block.

- Incoming connections
  The memory cell's input can receive outputs from the hidden layer and the input layer;

  - The gate units can receive from the outputs from the hidden layer, input layer and the value of internal states.
  - The output unit can only receive value from the outputs of conventional hidden units and memory cells.

By including the forget gate and peephole weights, the traditional LSTM evolved to the modern LSTM and the update scheme in Hochreiter et al. [1997] needs refinement.

Electronic copy available at: https://ssrn.com/abstract=3307566

```
for i = 1 to n do
    Initialize. Initialize all weights {w_ij(1)} randomly.
    while stopping criteria not met do
        for t = 1 to T − 1 do
            Feedforward from x_t to output y_t using weights {w_ij(t)}.
            Back propagatation. calculate error signals based on y_t − z_t. Take down |y_t − z_t|.
            Update weights. {w_ij(t)} → {w_ij(t + 1)}
        end for
        Reset network except for its weights.
    end while
end for
```

Figure 5.9: Pseudo-code of LSTM. The subscripts i and j standards for any units that have connections. y_t and z_t denote the actual output and target output of the network at time t. $z_t = x_{t+1}$. n is the number of simulations.

5.2.3 The learning algorithm

Details and proofs of the learning algorithm of the LSTM can be found in Hochreiter et al. [1997]. We display the learning algorithm in Figure 5.9. We emphasise here that the update scheme involves truncated derivative which are used to enhance the efficiency of the network. Note that the *training error* is the average of the $|z_t - y_t|$, $t = 1, \cdots, T$, in Figure 5.9. The weight update scheme is illustrated in Figure 5.10. The updated algorithm for the backward pass is a combination of truncated BPTT and customised RTRL. The former refers to BPTT using truncated derivatives. To be specific, standard BPTT is applied to output units, truncated BPTT is implemented by output gates, while weights related to input gates, forget gates and memory cells use a truncated RTRL. By comparing Figure 5.3 with Figure 5.10 we can get an idea of the differences between the full BPTT and the truncated one used by LSTM. In the BPTT algorithm for standard RNN, Equation (5.1.4) shows that the information of all time steps has to be saved for the update of weight and the weights remain unchanged during the entire feedforward process. Whereas the LSTM network updates its weights at each time step during the forward pass, so that there is no need to store all values. To summarise, the BPTT with full gradient change the weight for one time by passing forward from time $t = 1$ to $t = T$ and passing backward from $t = T$ to $t = 1$ while the truncated BPTT passes forward from $t = 1$ to $t = T$ and the weights are updated for T times.

5.2.3.1 Computational Complexity of LSTM

The LSTM algorithm is efficient and its computational complexity is of order $O(W)$ per time step, where W is the number of weights (see Hochreiter et al. [1997]). Compared with BPTT, LSTM is more economical in terms of space. In fact, calculating the full gradient of LSTM also has advantage since it is easy to debug and can be checked by numerical approximations (see Graves et al. [2005]). With the CEC, a LSTM block may be considered as a smart network unit, compared to the conventional hidden unit, to store information for arbitrary length of time. Therefore, the LSTM is well-suited to process and predict time series when there are unknown size of long time lags between important events. Since financial time series has long term memory, predicting volatility by LSTM network may be rather promising. While the traditional RNN with weight update algorithm BPTT (Back Propagation Through Time) (see Williams et al. [1990]) and RTRL (Real-time Recurrent Learning) (see Robinson et al. [1987] and Williams et al. [1992]) and the combinations of the former two (see Schmidhuber [1992]) have been proved to occur learning failure when processing sequences with only 10 time steps (see Bengio et al. [1994], Hochreiter et al. [1997], Gers et al. [2000], Hochreiter et al. HochreiterEtAl01), the LSTM can deal with 1000 time steps and even more, outperforming those traditional RNN algorithms. In fact, except for prediction, LSTM outperforms other RNNs in

Electronic copy available at: https://ssrn.com/abstract=3307566

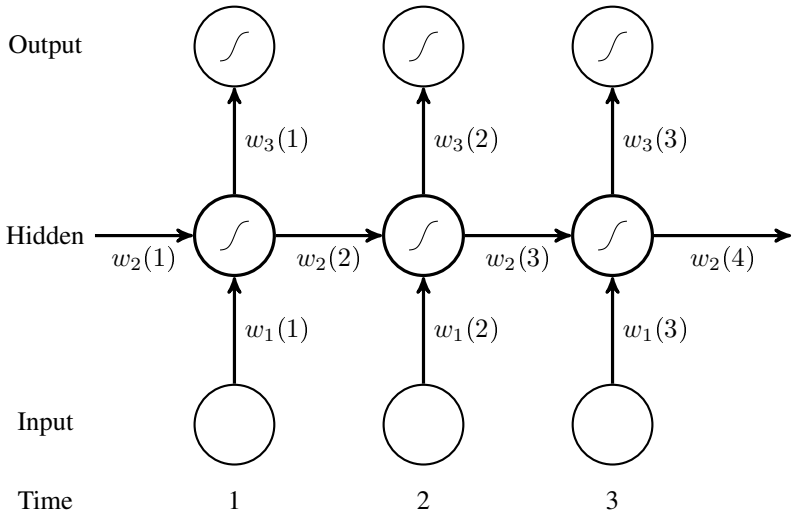


Figure 5.10: Algorithm of weight update of LSTM. Note that the memory cell is regarded as a normal hidden unit for simplicity. Only one hidden unit is shown.The number in bracket represents the time step.Weights are updated at each time step.

numerous aspects such as the best performance in speech recognition (see Graves et al. [2011]) and the ICDAR handwritting competition in 2009.

5.3 Reservoir computing

Reservoir Computing (RC) is a special RNN originating from Echo State Networks (ESNs) (see Jaeger [2001]) and Liquid State Machines (LSMs) (see Maass et al. [2002]), which assumes that supervised adaptation of all interconnection weights are not necessary, and only training a memoryless supervised readout from it is sufficient to obtain good results. Thus, it avoids the shortcomings of the gradient-descent training of RNNs. RC is based on the computational seperation between a dynamic reservoir and a recurence-free readout. The former is an RNN as a nonlinear temporal expansion function, randomly created and unchanged during training. The latter produces the desired output from the expansion. This is a very simple approach avoiding the problem of bifurcations encountered during the training of RNNs, and the use of complex memory cells when learning long-term dependencies, as in the LSTM described in Section (5.2). Even though several studies aimed at understanding RC and the factors affecting its performances, none have been completely satisfactory due to the complexity of the reservoir, often resulting in contradictory conclusions. Introduction to the concepts and methodologies can be found in the literature (see Lukosevicius et al. [2009], [2012]). Goudarzi et al. [2014] compared the performance of three methods, the delay line, the NARX network, and the ESN and concluded that the first two have higher memorisation capability, but fall short of the generalisation power of the latter. Thus, we are going to briefly describe the ESN.

5.3.1 Describing the Reservoir methods

5.3.1.1 Description

We let the time-varying input signal be an N_i-th order column vector $U(t) = [u_i(t)]$, the reservoir state is an N_x-th order column vector $X(t) = [x_j(t)]$, and the generated output is an N_o-th order column vector $Y(t) = [y_o(t)]$. A Reservoir Computer is a collection of internal nodes, whose state vector $X(t)$ evolve in discrete time t according to the nonlinear map in Equation (8.1.4) and given in the form

Electronic copy available at: https://ssrn.com/abstract=3307566

$$X(t+1) = (1-\alpha)X(t) + \alpha f\big(W_{res}^{\top} \cdot X(t) + W_{in}^{\top} \cdot U(t+1) + W_{fb}^{\top} \cdot Y(t)\big) \tag{5.3.8}$$

for some activation function $f(\cdot)$, where the leaky integrator α is a constant controlling the speed of the dynamics. The input weight matrix is an $N_i \times N_x$ matrix $W^{in} = [w_{ij}^{in}]$ where w_{ij}^{in} is the weight of the connection from input node i to reservoir node j. The connection weights inside the reservoir are represented by an $N_x \times N_x$ matrix $W_{res} = [w_{jk}^{res}]$ where w_{jk}^{res} is weight from node j to node k in the reservoir. In presence of a bias, the output matrix is an $(N_x+1)\times N_o$ matrix $W_{out} = [w_{ko}^{out}]$, where w_{ko}^{out} is the weight of the connection from the reservoir node k to the output node o. In the case where the output nodes are also connected to the input nodes and to themselves the size of the matrix becomes $(N_x + N_i + N_o) \times N_o$. The feedback matrix is an $N_x \times N_o$ matrix $W_{fb} = [w_{ko}^{fb}]$, where w_{ko}^{fb} is the weight of the connection from the reservoir node k to the output node o. If no output feedback is needed, then W_{fb} is null. The generated output is given by

$$Y(t) = f_{out}(W_{out}^{\top} \cdot X(t)) \tag{5.3.9}$$

where $f_{out}(\cdot)$ is an output activation function, typically the identity or a sigmoid. The output weights are trained to minimise the squared output error

$$E = ||Y(t) - \hat{Y}(t)||^2$$

given the target output $\hat{Y}(t)$. Once we know the optimum weights of the network, we can use the model to perform a forecast. To do so, we let T be a network state update, where

$$X(t+h) = T(X(t), \overline{U}^h)$$

denote the network state resulting from an iterated application of Equation (5.3.8) when the input sequence $\overline{U}^h = U(t+1), ..., U(t+h)$ is fed to the network being in state $X(t)$ at time t.
The basic steps to construct an ESN are as follows:

- $\mathbf{W}_{in}$ and $\mathbf{W}_{res}$ are randomly initialised, an appropriate α is selected.
- For each t,
  - Import input vector $\mathbf{U}(t)$ of the training set to the system and evolve the corresponding reservoir state $\mathbf{X}(t)$.
- A supervised learning algorithm is built on pairs of $\mathbf{X}(t)$ and $\hat{\mathbf{Y}}(t)$.
- For each t,
  - Apply trained $\mathbf{W}_{out}$ on input vector $\mathbf{U}(t)$ of test set to compute predictions $\mathbf{Y}(t)$.

5.3.1.2 The parameters

We present some of the major characteristics one has to consider when defining a relevant model (see Lukosevicius et al. [2009] and Lukosevicius [2012]). There are several features one has to consider when designing RC:

1. Connectivity: $\{W_{in}, W_{res}, W_{fb}\}$ is generated randomly. The input, bias, reservoir and feedback scaling (respectively named s_{in}, s_{bias}, s_{res} and s_{fb}) will be considered as micro parameters. For the choice of topology, the authors suggested a permutation matrix with a medium number and different lengths of connected cycles, or a general orthogonal matrix.
   (a) N_x: The principle is that the bigger the reservoir, the better the obtainable performance, provided appropriate regularisation measures are taken against over-fitting. As ESNs are computationally cheap, we can use as big a reservoir as we can afford computationally. N_x will be considered as a macro parameter.

Electronic copy available at: https://ssrn.com/abstract=3307566

(b) Sparsity: It enables fast reservoir updates. One rule of thumb is to connect at most each node to 10 other nodes in the reservoir.

(c) Distribution: Uniform or Normal. W_{in}, W_{res} and W_{fb} have usually the same distribution. In general, the seed of the pseudo-random generator is fixed.

2. Leaking Rate α: The leaking rate can be seen as the time interval between two discrete realisations. In general, we expect its value to be close to 1.

3. Echo State Property: The state of the reservoir $X(t)$ should be uniquely defined by the fading history of the input $U(t)$ [...] The spectral radius should be greater in tasks requiring longer memory of the input. Finding clear characteristics remain a large debate among practitioners and researchers.

From these features, one can see that many meta parameters are involved, so that it is necessary to perform an additional optimisation aiming at minimising the RMSE on the validation sample. It is known as meta-optimisation (see Pedersen et al. [2008a]). The idea is to have an optimisation method act as an overlaying meta-optimiser, trying to find the best performing behavioural parameters for another optimisation method (see Pedersen et al. [2008a]). Note, parameter tuning via meta-optimiser is done in an offline manner, while adaptation of DE parameters is done online during optimisation.
According to Lukosevicius et al. [2009], the investigation shows that the error surfaces in the combined global parameter and W_{out} spaces may have very high curvature and multiple local minima. Thus, gradient descent methods are not always practical. In general, optimising reservoir is quite a hard task as we have many parameters involved. Nevertheless, the computationally cheap property of Reservoir Computing leads to the following idea: *"generate k reservoirs and pick the best."*.

5.3.1.3 The readout

Echo State Networks (ENSs) assume that if a random RNN possesses certain algebriac properties, it suffises to train a linear readout from it to obtain good performances. Even though there are several readout methods, the most popular one is the linear regression where W_{out} solve a system of linear equations. From Remark (**??**), we get

$$W_{out}^{\top} X_m = \hat{Y}_m$$

where both the matrix $X_m \in \mathbb{R}^{N_x \times T}$ and the matrix $\hat{Y}_m \in \mathbb{R}^{N_o \times T}$ [1], over the training period $t = 1, ..., T$, have a column for every training time step t. The output weights can be estimated with direct pseudo-inverse calculations, or they can be estimated with the ordinary linear regression (Wiener-Hopf solution). Thus, we get

$$W_{out}^{\top} X_m \cdot X_m^{\top} = \hat{Y}_m \cdot X_m^{\top}$$

and the weight matrix becomes

$$W_{out}^{\top} = \hat{Y}_m \cdot X_m^{\top} (X_m \cdot X_m^{\top})^{-1} = PR^{-1}$$

where $R = X_m \cdot X_m^{\top}$ is the correlation matrix of the reservoir states, and $P = \hat{Y}_m \cdot X_m^{\top}$ is the cross-correlation matrix between the states and the desired outputs. Note, $P \in \mathbb{R}^{N_o \times N_x}$ and $R \in \mathbb{R}^{N_x \times N_x}$ do not depend on the training length T and can be calculated incrementally. When R is ill-conditioned the method is numerically unstable, but computing the Moore-Penrose pseudo-inverse R^{+} instead of R can improve the solution. Alternatively, we can decompose the matrix R into two triangular matrices with Cholesky or the LU decomposition (see Press et al. [1992]) and solve

$$W_{out}^{\top} R = P$$

[1] These matrices are transposed compared to the conventional notation.

Electronic copy available at: https://ssrn.com/abstract=3307566

by two steps of substitution to get $W_{out}^{\top} = PR^{-1}$.
Evolutionary search can also be used for training the linear readouts. There exists several bias to reduce the error in the validation set. One can smooth the model with regularisation functions with the Ridge regression

$$W_{out}^{\top} = P(R + \beta^2 I)^{-1} \tag{5.3.10}$$

where β controls the smoothing effect, and $I \in \mathbb{R}^{N_x \times N_x}$ is the identity matrix.

The following algorithm sums up the steps involved in the training of the ESN framework.

Algorithm 8 RC offline Training

Require: get a linear system solver (we use Cholesky Decomposition).
Require: set t_0 as a wash-out size.
Require: set β as a regularisation parameter.
1: $Y \leftarrow [y(t_0) \| \dots \| y(T)]$
2: $Y \leftarrow f_{out}^{-1}(Y)$
3: $X \leftarrow [1 \| U(t_0) \| \dots \| U(T) \| X(t_0) \| \dots \| X(T)]$
4: $W_{out} \leftarrow Y \cdot X^T \cdot (X \cdot X^T + \beta^2 I)^{-1}$
5: **return** W_{out}

Note, we can also add noise $\epsilon(t)$ (sampled from uniform or Gaussian distribution) to the reservoir states

$$X(t+1) = (1-\alpha)X(t) + \alpha f\big(W_{res}^{\top} \cdot X(t) + W_{in}^{\top} \cdot U(t+1) + W_{fb}^{\top} \cdot Y(t)\big) + \epsilon(t)$$

to stabilise solutions in the model (see Jaeger [2007]). We can further adjust global control parameters to make the echo state network dynamically similar to the system we model. For instance, we can use fully connected reservoirs or sparsely connected ones.

5.3.1.4 Some comments

In order to produce a reservoir with a rich enough set of dynamics the number of internal connections N_x should be large, the weight matrix W should be sparse, and the weights of the connections should be generated randomly from a uniform distribution symmetric around the zero value. Further, the network should have the echo state property (ESP), which relates asymptotic properties of the excited reservoir dynamics to the driving signal (see Jaeger [2001]). It states that the effect of a previous state $X(t)$ and a previous input $U(t)$ on a future state $X(t+h)$ should vanish gradually as $h \to \infty$, and not persist or be amplified. In reservoirs using the $tanh$ squashing function, and for zero input, the reservoir weight matrix W^{res} must be scaled so that its spectral radius [2] $\rho(W^{res})$ satisfies $\rho(W^{res}) < 1$. For any kind of inputs (including zero) and state vectors, we require $\sigma_{max}(W^{res}) < 1$ where $\sigma_{max}(W^{res})$ is the largest singular value of W^{res}. If the input comes from a stationary source, the ESP holds with probability 1 or 0. Due to the auto-feedback nature of RNNs, the reservoir states $X(t)$ reflect traces of the past input history, which can be seen as a dynamical short-term memory. Assuming a single input ESN, the short-term capacity is given by

$$C = \sum_i r^2(U(t-i), Y_i(t))$$

where $r^2(\cdot,\cdot)$ is the squared correlation coefficient between the input signal delayed by i and an output signal $Y_i(t)$ trained to memorise $U(t-i)$ on the input signal $U(t)$. For an i.i.d. input, the memory capacity C of an echo state network of size N is bounded by N. Thus, we can not train ESN on tasks requiring unbounded-time memory.

[2] the largest absolute eigenvalue

Electronic copy available at: https://ssrn.com/abstract=3307566

5.3.2 Choosing the parameters

Choosing the right parameters can be more beneficial than choosing the right model itself. However, it is never trivial to tune parameters of ESN. Experience will play an important role here. It is inevitable and necessary to use manual selection to get a sense of which parameters affect results more than others do, and these manually selected values could be used as the initial values for further automated selection. During the process, ensure that *only* one parameter is changed at a time. The quality of a particular parameter setting is evaluated by using root mean square error of training set and test set. Note that all the parameters ought to be optimised under a condition where adequate regularisation measure is taken to alleviate overfitting.
Practically, the input scaling, leak rate and spectral radius of reservoir weight matrix are the three most important parameters of ESN.

5.3.2.1 Input Scaling

Input weight matrix $\mathbf{W}_{in}$ is usually dense without zero elements. The entries of the matrix usually follow a uniform distribution in a symmetric range $[-a, a]$ or classic range $[0, 1]$. Often candidate values for a is 1, 0.1, 0.01. The distribution can be Gaussian or a customised one as well, whereas the boundness is not guaranteed.
The first column of the input weight matrix is usually the bias column, which corresponds to bias element $'1'$ in input vector $\mathbf{U}(t)$. Hence it does not share the same status as the rest of the columns. Thus, to reduce degree of freedom, all columns except the first column of the input weight matrix are randomized using one parameter a, and the first column uses another parameter b.
Small values of a will map input to a small range around zero, where $tanh(\cdot)$ is virtually linear, hence this setting is suitable for solving linear systems. On the other hand, large values of a will map $\mathbf{u}(n)$ to extreme values of $tanh(\cdot)$, namely -1 or 1, creating a non-linear, binary manner. Overall, input scaling controls non linearity of $\mathbf{X}(t)$.

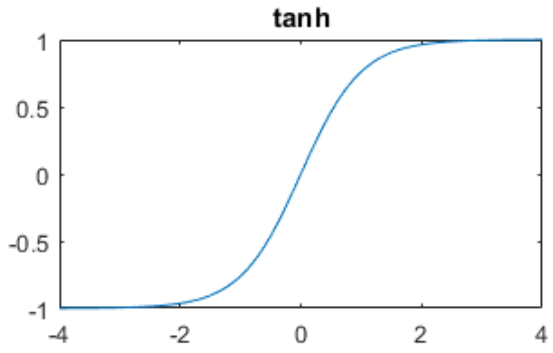


Figure 5.11: activation function tanh

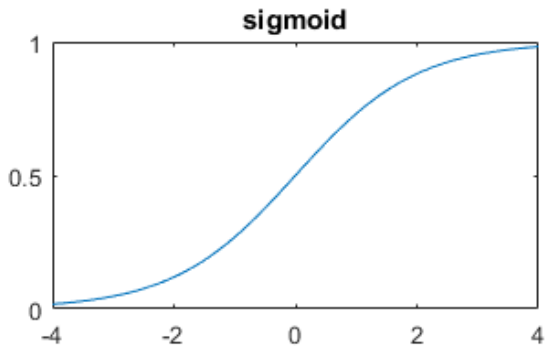


Figure 5.12: activation function sigmoid

The reservoir weight matrix $\mathbf{W}_{res}$ represents a structure of recurrently connected units. Similar to the case of $\mathbf{W}_{in}$, non-zero elements of $\mathbf{W}_{res}$ usually have uniform distribution in a symmetric range. However, unlike $\mathbf{W}_{in}$, $\mathbf{W}_{res}$ is sparse and has other distinct properties.

5.3.2.2 Size of the reservoir

Intuitively, larger the reservoir, the better the attainable performance. the size can be as big as 20, 50, 100, 1000, even 10000 is not uncommon. There is few condition when reservoir may be oversized: one is that the task itself is pretty much trivial, and the other could be that the size approaches meaningful fraction of number of data points, or in other

Electronic copy available at: https://ssrn.com/abstract=3307566

words, lack of data points. Normally, if sample size is n, it is required that (see Jaeger [2002])

$$N_x <= n/2; \tag{5.3.11}$$

Typically, it also holds that $1 + N_u + N_x << n$, this constraint also reduces the risk of overfitting. At the same time, the size of reservoir should at least be the number of independent time steps that are supposed to be memorised to address the problem.

5.3.2.3 Sparsity

Though the reservoir is big, the reservoir weight matrix is always set to be *sparse* in many publications. In general, higher density tends to give higher prediction performance. However, at the same time, lower sparsity means more recurrent connections, and large non-zero matrix operations increase computational costs. Generally, above a certain threshold, connectivity barely affects the final performance. Hence, one can recommend 30% connectivity, which is robust enough to be a balance between efficiency and efficacy. Compared to others, sparsity has relatively low priority to be optimised. In Matlab, there is a special representation for sparse matrix that could speed up computations.

5.3.2.4 Spectral Radius

Being the central parameter of ESN, spectral radius is the maximum of absolute eigenvalues of reservoir matrix $\mathbf{W}$. Intuitively, it determines the width of the distribution of non-zero entries. If it is set too high, then the reservoir may hold periodic, even chaotic spontaneous attractor modes which deactivates echo state property. It has been proved that, in most cases, $r(\mathbf{W}) < 1$ ensures the echo state property (see Lukosevicius [2012]). However, it is also true that $\mathbf{W}$ with spectral radius greater than 1 retains echo state property. As a rule of thumb, if the task requires extensive input history, then it is recommended to set the spectral radius higher. The following steps are usually taken when specific spectral radius is assigned to reservoir weight matrix:

1. Initialize a reservoir weight matrix $\mathbf{W}_0$
2. Normalize $\mathbf{W}_0$ to $\mathbf{W}_1$ using $\mathbf{W}_1 = \mathbf{W}_0/r_1$, where r_1 is the spectral radius of $\mathbf{W}_0$. Now that $\mathbf{W}_1$ has unitary spectral radius.
3. Multiply $\mathbf{W}_1$ by r element-wise so that it has spectral radius r.

5.3.2.5 Leak Rate

Leak rate is a real value between 0 and 1. Echo state network without leak rate α is called standard echo state network, the one with leak rate or leaky integrator is called LI-ESN. In this latter case, α can either be placed before application of $f(\cdot)$ or after. Introduction of leak rate makes it possible to learn very slow dynamic systems and replay them at various speeds (see Lukosevicius et al. [2009]).
Small values of leak rate may induce slow dynamics of $\mathbf{X}(t)$ and extend short term memory (see Jaeger [2012]). Virtually, the prediction performance is sensitive to leak rate since a small change in leak rate may lead to a big change in final result. Hence, this parameter also has relatively high priority. Some low-prioritised parameters could be assigned default values for convenience.

5.3.3 Some improvements

Separating the reservoir from the readout training allows for two research directions to be pursued independently,

1. the generation of the reservoir, and
2. the output training.

Electronic copy available at: https://ssrn.com/abstract=3307566

There is no reasons why the reservoir should be randomly generated and alternative methods could be used to obtain optimal reservoir design. However, no single type of reservoir can be optimal for all types of problems (no free lunch principle). Several methods have been proposed for generating the reservoir (see Lukosevicius et al. [2009]), which can be classified as

1. generic methods for generating $RNNs$ with different neuron models, connectivity patterns and dynamics.
2. unsupervised adaptation of the reservoir based on the input data $U(t)$ but not the target value $\widehat{Y}(t)$.
3. supervised learning, adaptation of the reservoir using task-specific information from both $U(t)$ and $\widehat{Y}(t)$.

In order to deal with different time scales simultaneously, one can divide the reservoir into decoupled sub-reservoirs and introduce inhibitory connections among all the sub-reservoirs. However, the inhibitory connections should be heuristically computed from the rest of W and W_{fb} such that they predict the activations of the sub-reservoirs one time step ahead (see Xu et al. [2007]). Alternatively, the Evolvino transfers the idea of ESNs to a LSTM type of RNNs where the LSTM RNN used for its reservoir consists of specific small memory-holding modules. In that model, the weights of the reservoir are trained using evolutionary methods (see Schmidhuber et al. [2007]). Further, ESNs like any other RNNs has only a single layer of neurons (see Figure 5.1a), making it unsuitable for some types of problems requiering multilayers. A solution is to use Layered ESNs (see Lukosevicius [2007]), where part of the reservoir connections are instantaneous, and the rest takes one time step for the signals to propagate as in normal ESNs. One can also add leaky integrator neurons to ESNs, getting leaky integrator ESNs (Li-ESNs) performing at least as well as the simple ESN (see Lukosevicius et al. [2006]). Since the parameters a and Δt control the speed of the reservoir dynamics, small values result in reservoirs reacting slowly to the input. Note, depending on the speed at which the input $U(t)$ changes, we can vary Δt on-the-fly, getting a warping invariant ESNs (TWIESNs). Since checking the performance of a resulting ESN is relatively inexpensive, evolutionary methods for pre-training the reservoir developed (see Ishii et al. [2004]). Generally, one separate the topology and weight sizes of W_{res} to reduce the search space (see Bush et al. [2005]). Jiang et al. [2008] showed that by only adapting the slopes of the reservoir unit activation functions $f(\cdot)$ with an evolutionary algorithm, and having W_{out} random and fixed, a very good prediction performance of an ESN could be achieved. Note, evolutionary algorithms can also be used to train the readouts. One can increase the expressiveness of the ESN by having k linear readouts trained and an online switching mechanism among them, or by averaging outputs over several instances of ESNs (see Bush et al. [2006]).

5.3.4 Empirical results

It is generally argued that ESN is expert in fitting and predicting chaotic time series. This could be verified by assessing its performance on the classic Mackey-Glass series (see Figure (5.13)). We also verify performance of the models on denoised financial time series. The modelling is under the assumption that historical price series contains all necessary information to be used to predict next day prices and returns. We let the Linear Regression (LR) model be the benchmark (see Section (8.3.1.3)). We reproduce below some results on the Mackey-Glass signal obtained by a student of ours who did his MSc thesis at QFL (see Wang [2017]).

Electronic copy available at: https://ssrn.com/abstract=3307566

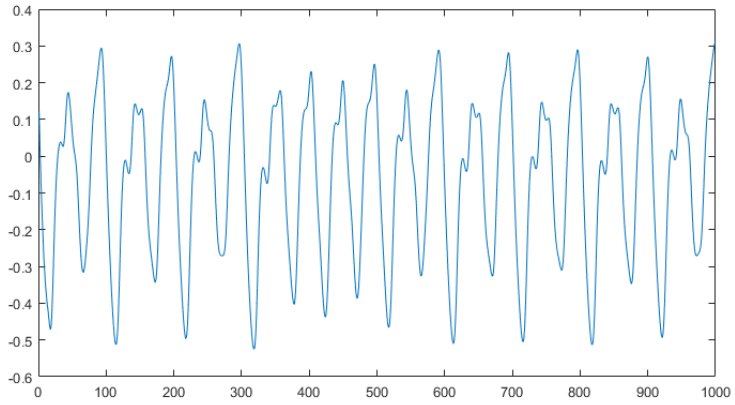


Figure 5.13: Samples of Mackey-Glass Chaotic Time Series

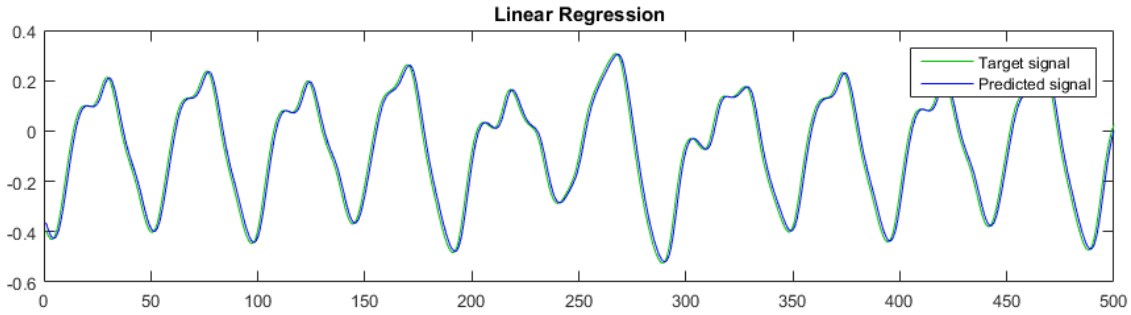
In a publication by Jaeger, it is claimed that advanced ESN could yield root mean square error (RMSE) of order 10^{-7} on predicting Mackey series (see Jaeger et al. [2004]).
We try to reproduce the experiment, using 2000 data points for training, with the first one hundred reservoir states $\mathbf{X}(t)$ being discarded due to initial transient effect, and 500 data points are used for testing. Both linear regression (LR) and ESN will be implemented. For ESN, parameters are specified as follows:

Leak Rate α	0.9	Spectral Radius	1.25
Input Weight Matrix $\mathbf{W}_{in}$	$[-0.5, 0.5]$	Reservoir Weight Matrix $\mathbf{W}_{res}$	$[-0.5, 0.5]$
Reservoir Size $\mathbf{N}_x$	400	Normalization Range	$[-1, 1]$
Reservoir Density	0.3	Regularization term β	10^{-8}

After plugging all the parameters into ESN, following errors are summarised:

	Training error(RMS)	Testing error(RMS)
Linear Regression	0.0315	0.032
ESN	7.9×10^{-6}	8.0×10^{-6}

Apparently ESN produces negligible errors compared to that of linear regression. In fact, even if reservoir size N_x is reduced to 35, training and testing errors are still less than 0.001, which is less than 3% of the errors made by linear regression.
For illustration, the predicting performance of ESN with $N_x = 35$, along with that of linear regression, are displayed below:



Electronic copy available at: https://ssrn.com/abstract=3307566

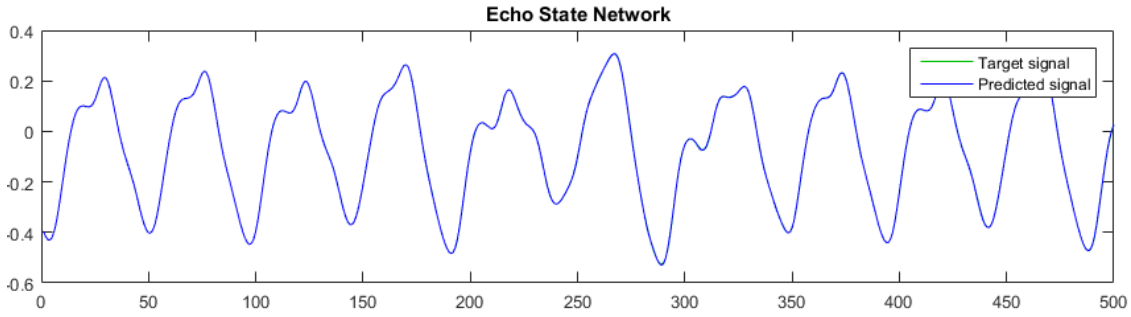


Figure 5.14: Mackey Series test set

In both figures, one could not virtually discern between predicted values and actual values, since the two curves nearly coincide with each other most of the time. Judging from the plot, it seems that linear regression has reasonably well fitted the data even if it has 'huge' test error compared with ESN. However, if we study more carefully the two figures, we can observe differences:

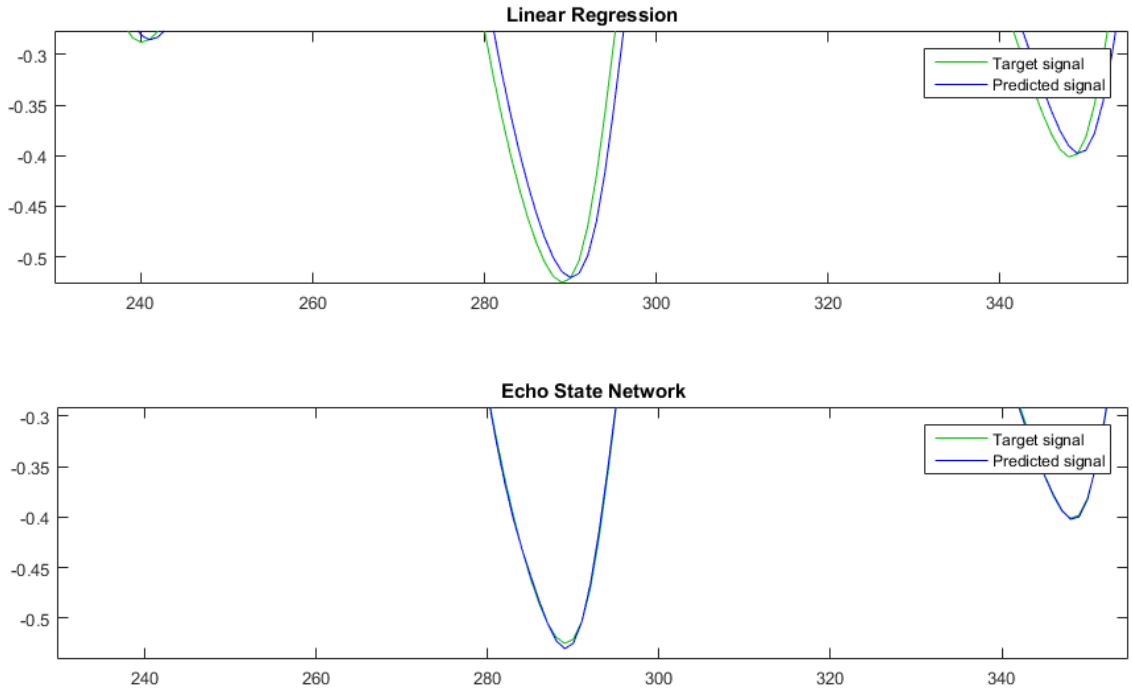


Figure 5.15: Zoomed-in Figure Comparison

One may notice that the prediction curve of the linear regression is just a forward shift of the target curves. In other words, the prediction curve is just a *time-lagged* copy of the actual target curve if horizontal axis is discrete time. Therefore, in this case, the prediction capability of linear regression is relatively poor. On the other hand, the ESN prediction curve tends to move *in phase* with the target curve. Even at the turning point around $t = 290$, where the two curves can visually be distinguished, the trends of prediction values keep pace with that of target curves. Hence, statistically, ESN has much better test error than linear regression. Below are some samples of reservoir states $\mathbf{X}(t)$:

Electronic copy available at: https://ssrn.com/abstract=3307566

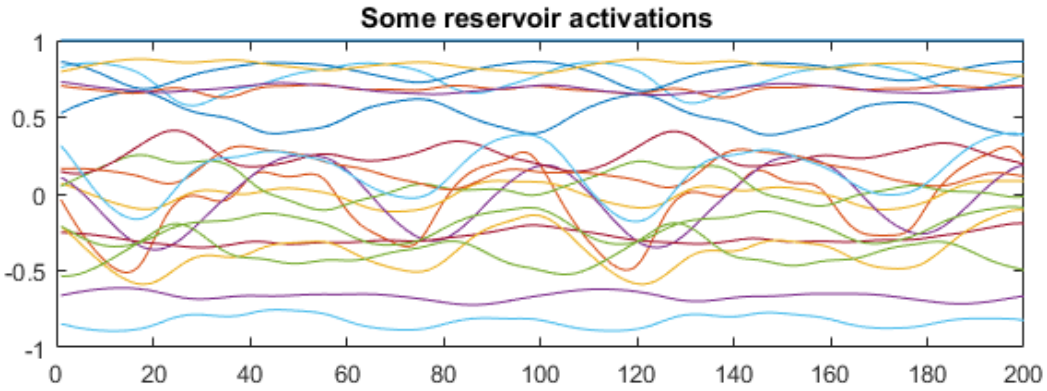


Figure 5.16: Evolutions of reservoir activations

In the figure above, reservoir activations actually means a vertical concatenation of bias factor, input vector and actual reservoir state, that is, $(1, \mathbf{U}(t), \mathbf{X}(t))^\top$. It is a vector of size $1 + N_u + N_x$, since N_x is large, the first 20 entries of the vector are plotted. Hence in the graph there are a total of 20 curves, with each curve being the evolutions of a fixed position within the vector $(1, \mathbf{U}(t), \mathbf{X}(t))^\top$. At the same time, the x-axis indicates $t = 1$ to 200, and the number of evolutions is 200. In the figure, the reservoir state is not saturated since not all curves converges to the same number(for example 1 or -1). Note also that the upper most curve, the bias factor, is a straight line which always equals to one.

The following is a bar plot of output weight matrix $\mathbf{W}_{out}$:

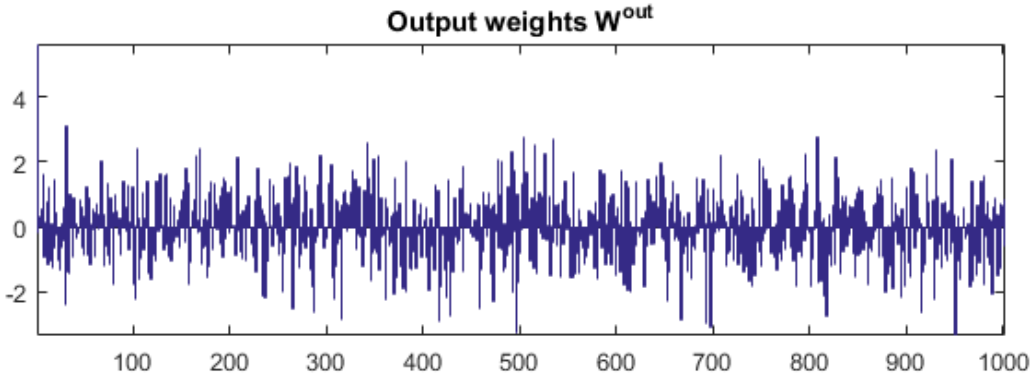


Figure 5.17: Evolutions of reservoir states

Assuming $N_y = 1$, $\mathbf{W}_{out}$ is a vector of size $N_x \times 1$. In the figure above, $N_x = 1000$, and almost all the elements of $\mathbf{W}_{out}$ are within the range $[-3, 3]$. Hence, there are no extreme large values in $\mathbf{W}_{out}$ and the model is less likely to suffer from overfitting.

Recall that the word 'chaotic' means that a time series consists of irregular patterns, so that visually no obvious trends could be easily perceived, while 'noisy' emphasises on stochastic or irrelevant factors that may interfere or even obscure useful inputs. A time series could be both chaotic and smooth at the same time, e.g. Mackey Series, while noisy time series are usually filled with *wavelets*.

Electronic copy available at: https://ssrn.com/abstract=3307566

After adding some Gaussian noise to the standard Mackey series, the following figure is plotted:

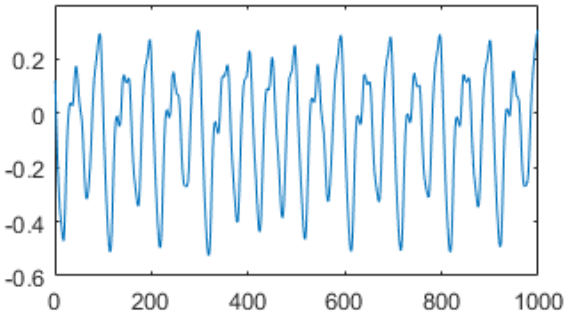


Figure 5.18: Mackey series

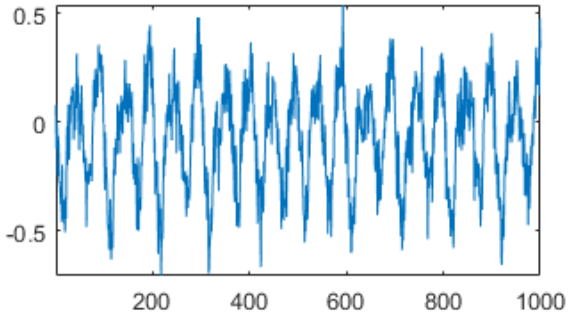


Figure 5.19: noisy Mackey series

As in figure 5.19, the noisy curves are usually oscillating more frequently and therefore are much more difficult for any models to track its path. As a matter of fact, it could be seen that ESN possesses advantages in fitting and predicting chaotic time series. However, experiments show that the performance of ESN will be discounted coping with noisy time series directly.

Unfortunately, financial time series is often chaotic and noisy. In addition, great caution is required when financial data are used for backtesting. There is no doubt that flawed data can lead to misleading results. In worst cases, historical returns may be overestimated. Following are several facts about financial data:

- Dataset with high resolution offers more details, at the expense of being noisier (Aamodt, 2015, S. 50)
- Stock prices need to be adjusted after share split or dividend pay-out.
- Most importantly, many accessible datasets have survisorship bias (see Chan [2009]) since historical database does not hold information about companies which disappeared due to delistings, bankruptcies or mergers.

Along with the first two bullet points, the size of dataset also matters in a machine learning model, while the third bullet point, as well as transaction cost, is crucial in terms of backtesting.

Due to limited public access to all types of financial datasets, the focus is on modelling stock returns. Hence, regardless of survivorship bias, only survived stocks, especially those from S&P 500, are chosen for experiments. Generally, these stocks have some advantages over other stocks in terms of market cap and liquidity. The index S&P 500 itself is often used as the weather report of financial industry.

In our experiment, 100 constituents of S&P 500 are imported. For each stock, the daily closing price data are downloaded. 1200 data points are used for training (the first 100 points are used to evolve reservoir states $\mathbf{X}(t)$ and will be discarded), while another 200 points are left out for testing.

Applying linear regression to the raw log-returns of symbol MMM gives the following figures:

Electronic copy available at: https://ssrn.com/abstract=3307566

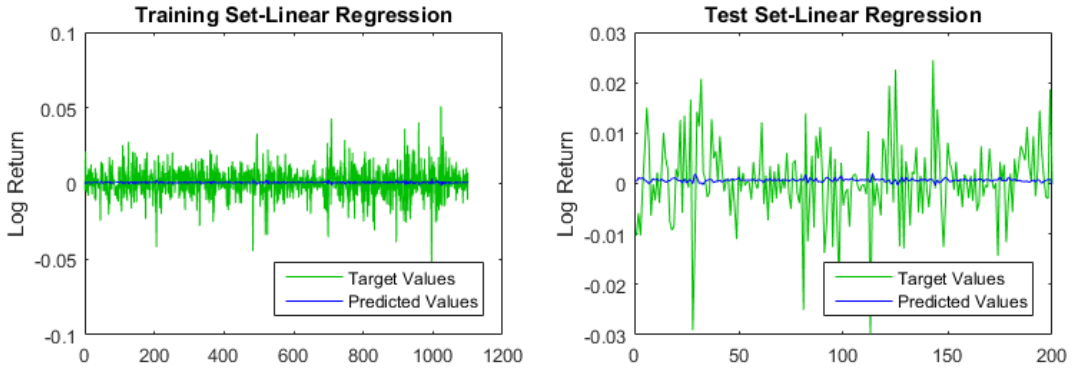


Figure 5.20: Linear Regression on raw stock returns

The figure shows that the log-return time series is so intractable and noisy that only approximately a straight line around zero would minimise the least square error for linear regression, where the regularisation term β is the only parameter of ridge regression. However, regardless of what β is, it always gives similar results. Here, the ratio of predicted values volatility to target values volatility would be close to zero, which is undesirable.

Now, the same raw data are fed into the ESN:

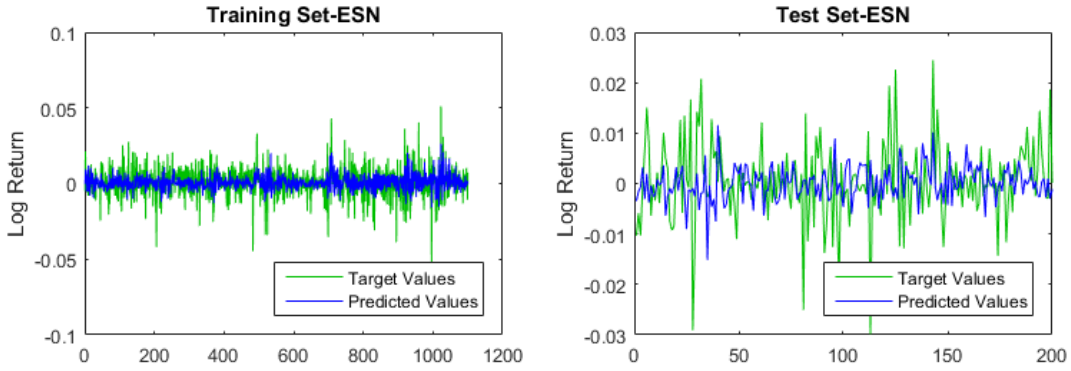


Figure 5.21: ESN on raw stock returns

Statistically, the ESN has test error of only 5% lower than that of linear regression, despite the fact that visually their figures differ a lot. Moreover, in dealing with unprocessed data, adjustment of parameter values does not improve much the performance of the ESN.

Now, the log-return series of MMM is denoised before fed into linear regression and ESN. Choosing the regularisation term $\beta = 10^{-8}$, the linear regression gives the following error:

	Training error(RMS)	Test error(RMS)
Linear Regression	0.00472	0.0036

At the same time, the ESN is equipped with common parameter settings below,

Leak Rate α	1	Spectral Radius	0.8
Input Weight Matrix $\mathbf{W}_{in}$	$[-2, 2]$	Reservoir Weight Matrix $\mathbf{W}_{res}$	$[-0.5, 0.5]$
Reservoir Density	0.5	Regularisation Term β	10^{-8}

Electronic copy available at: https://ssrn.com/abstract=3307566

In Matlab, the same random initialisations for $\mathbf{W}_{in}$ and $\mathbf{W}_{res}$ are reproducible by setting values for seed. For example, rand('seed', 34);

Now, fix these parameters, and only vary the reservoir size N_x since it can sensitively affect performance of ESN. The results are displayed in triplets, which stands for N_x, training error and test error respectively.

ESN	f=tanh	f=sigmoid
No normalisation	$850, 0.00227, 0.00157$	$200, 0.00235, 0.00204$
Normalised to $[-1, 1]$	$170, 0.00209, 0.00150$	$600, 0.00228, 0.00140$
Normalised to $[-2, 2]$	$140, 0.00234, 0.00166$	$80, 0.00232, 0.00139$
Normalised to $[0, 1]$	$190, 0.00193, 0.00136$	$650, 0.00250, 0.00155$

From the table, it could be seen that:

- Normalisation generally improves task performance of ESN (note that normalisation does not virtually change performance of linear regression).
- Compared to tanh, sigmoid function works better in symmetric ranges. Especially with the range $[-2, 2]$, ESN with sigmoid activation function uses small reservoir size $N_x = 80$ and still perform better than that of tanh with a larger reservoir size $N_x = 140$.
- The best test error comes from ESN with normalisation range $[0, 1]$, it reduces 62% of the test error made by linear regression. Moreover, it is also coupled with best training error in the table.

Cautiously note that these are observations from the test on stock MMM. Further experiments on large set of stocks are required to check if the above observations can be generalised.

The corresponding figures for ESN with normalisation range $[0, 1]$ and linear regression are displayed below:

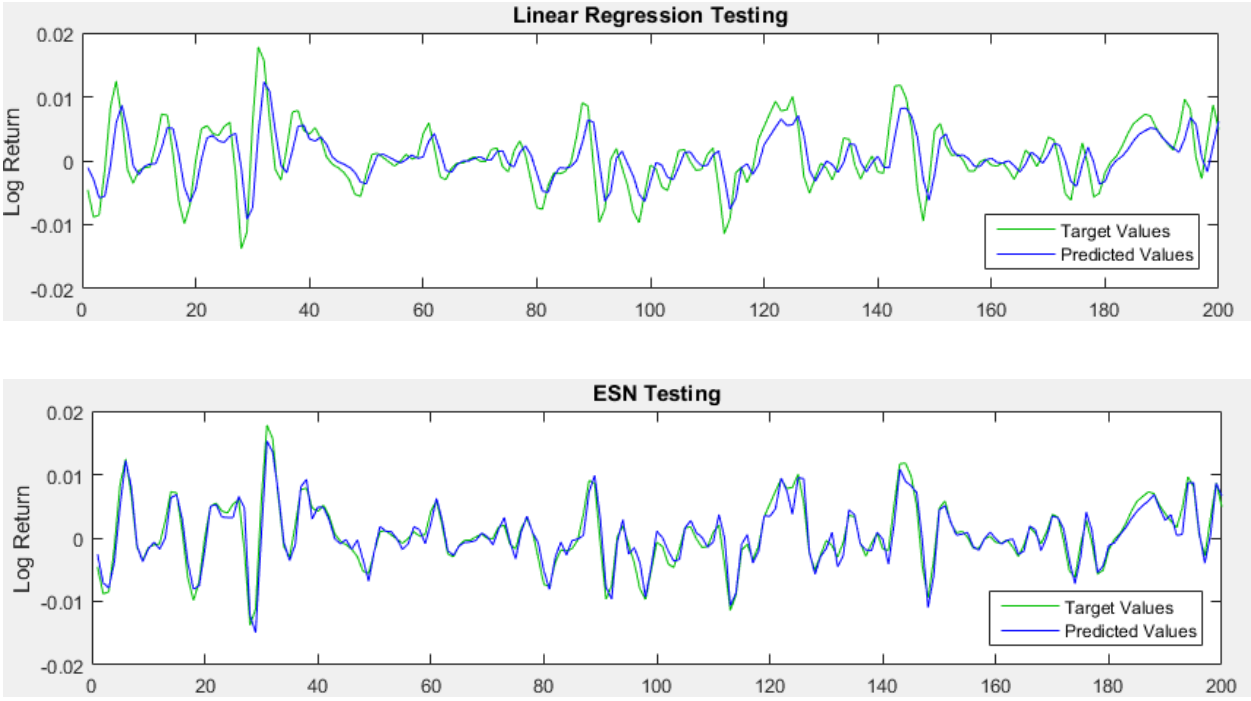


Figure 5.22: ESN on denoised stock returns test set

Electronic copy available at: https://ssrn.com/abstract=3307566

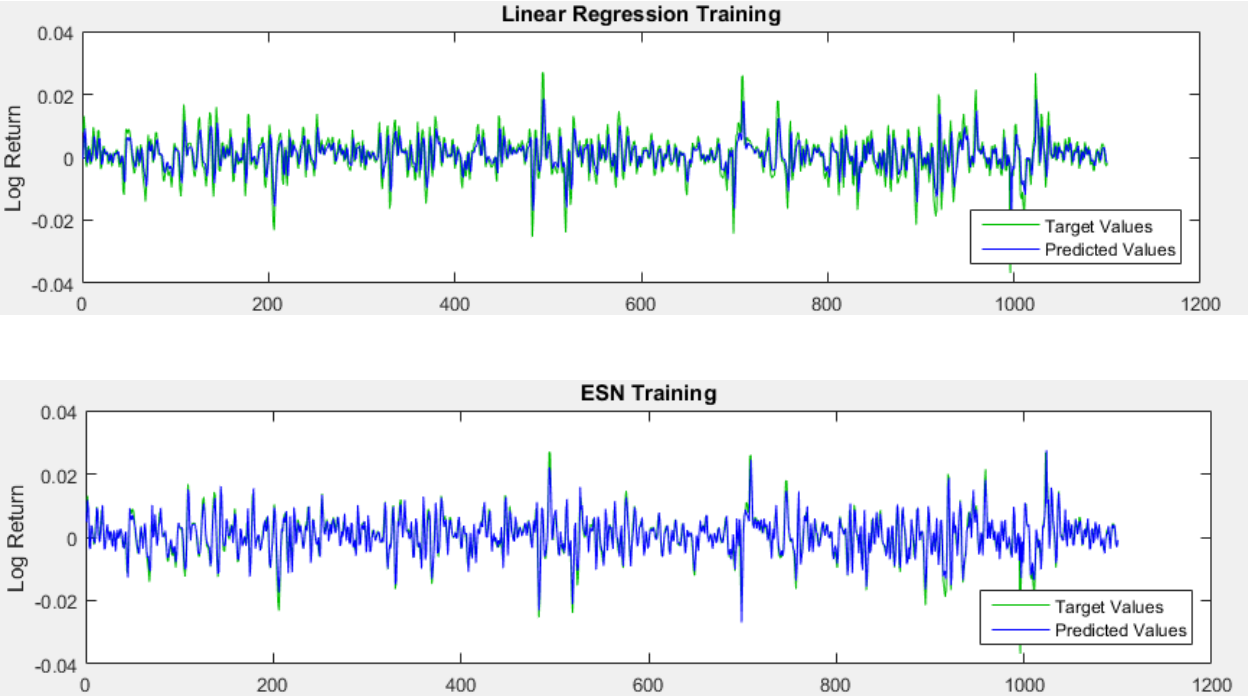


Figure 5.23: ESN on denoised stock returns training set

From these figures it can be seen that ESN restores some of its prediction capability when dealing with denoised chaotic time series instead of raw time series. ESN apparently has better fittings in comparison of linear regression. Statistically, with normalisation, ESN reduces at least 57% of the test error made by linear regression.
Moreover, note that parameters are manually tuned one at a time. Therefore, if a systematic way of tuning (that is, grid search, genetic algorithm) is used, the errors are likely to be further reduced by a meaningful percentage. Moreover, optimal parameters settings are specific and unique for each stock. If the same setting is used for various stocks, the corresponding errors would be inevitably increased. Nevertheless, an experiment is done with the following parameter setting:

Leak Rate α	1	Spectral Radius	0.8
Input Weight Matrix $\mathbf{W}_{in}$	$[-2, 2]$	Reservoir Weight Matrix $\mathbf{W}_{ref}$	$[-0.5, 0.5]$
Reservoir Density	0.5	Reservoir Size N_x	350

Meanwhile, a relative large regularisation term $\beta = 0.01$ is chosen to reduce chance of overfitting. Also, the sigmoid function with normalisation range $[-2, 2]$ is used. ESN with this parameter setting is applied on 100 stocks from S&P 500 and then gives the error table as follows:

	Average Training error(RMS)	Average Test error(RMS)
Linear Regression	0.00809	0.00799
ESN	0.00498	0.00515

This time the table shows that ESN reduces the error by only 36% instead of 57% of the test error made by linear regression. This is due to the use of common parameter setting for all 100 stocks for convenience. Nevertheless, 36% reduction still reveals competitive prediction capability of ESN across stocks with a common parameter setting.

Electronic copy available at: https://ssrn.com/abstract=3307566

5.4 Other Models

5.4.1 Variable weights neural networks

In conventional neural networks, connection weights are fixed. That is, the characteristics of the neural network is invariant (the invariant type neural network). The adaptive ability of neural networks to an unknown environment depends on its generalisation capability. However, there exists a limit of generalisation ability in invariant type neural networks. Yasuda et al. [2006] proposed the neural network with variable connection weights, which changes its connection weights and its characteristic according to the changing environment. Lam et al. [2014] presented the variable weight neural network (VWNN), allowing its weights to be changed in operation according to the characteristic of the network inputs.

5.4.1.1 Description

We are now going to briefly describe the VWNN. It consists of two traditional neural networks, namely tuned and tuning neural networks. The former is the one which actually classifies the input data, while the latter provides the weights to the tuned neural network according to the characteristic of the input data. The VWNN works on the principle that the connection weights are function to the external input signal.
As an example, we consider a three-layer feed-forward fully-connected neural network (denoted fixed model) with n_{in} inputs and n_{out} outputs, weights $\omega_{ji}^{(k)}$ and bias terms $b_j^{(k)}$, $k = 1, 2$. The input vector is $x(t) = [x_1(t), ..., x_{n_{in}}(t)]$ and the output vector is $y(t) = [y_1(t), ..., y_{n_{out}}(t)]$ (see Bryson et al. [1969]). We are going to modify that network to obtain its associated VWNN. To do so, we need the tuning neural networks NN_k, $k = 1, 2$, which provide connection weights $\omega_{ji}^{(k)}$, and bias terms $b_j^{(k)}$ to the fixed model. The tuning networks rely on the modified input vector $x^{'}(t)$, which consists of some selected features from $x(t)$. Given a predetermined constant selection matrix $S \in \mathbb{R}^{n^{'}_{n_{in}} \times n_{n_{in}}}$, then

$$x^{'}_{n_{in}}(t) = S \cdot x_{n_{in}}(t)$$

where $n^{'}_{n_{in}} \leqslant n_{n_{in}}$. For example

$$x^{'}_{n_{in}}(t) = \begin{bmatrix} x^{'}_1(t) \\ x^{'}_2(t) \end{bmatrix}, x_{n_{in}}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} \text{ and } S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

such that $x^{'}_{n_{in}}(t)$ selects $x_1(t)$ and $x_3(t)$ as the input of the tuning neural networks NN_k. The latter will produce output weight vector $W(t)$ consisting of all connection weights of the tuned neural network (fixed model). Eventually, the tuned neural network will then use $W(t)$ to process the input $x(t)$ and produce the output $y(t)$. See Figure (5.24) and Figure (5.25).

5.4.1.2 Application to finance

Clearly, the role of the matrix S is to reduce the dimensionality of the input signal x in view of generating dynamical weights as a function of the latter.
The method was applied to large size neural network. In the case of financial time series, we can let the matrix S be a lowpass filter or other denoising technique, so that the modified input $x^{'}$ of the tuning network becomes the filtered time series. This way, the fixed weights of the tuning network are associated with a smooth signal and the variable weights of the tuned network become a function of the filtered market returns.

Electronic copy available at: https://ssrn.com/abstract=3307566

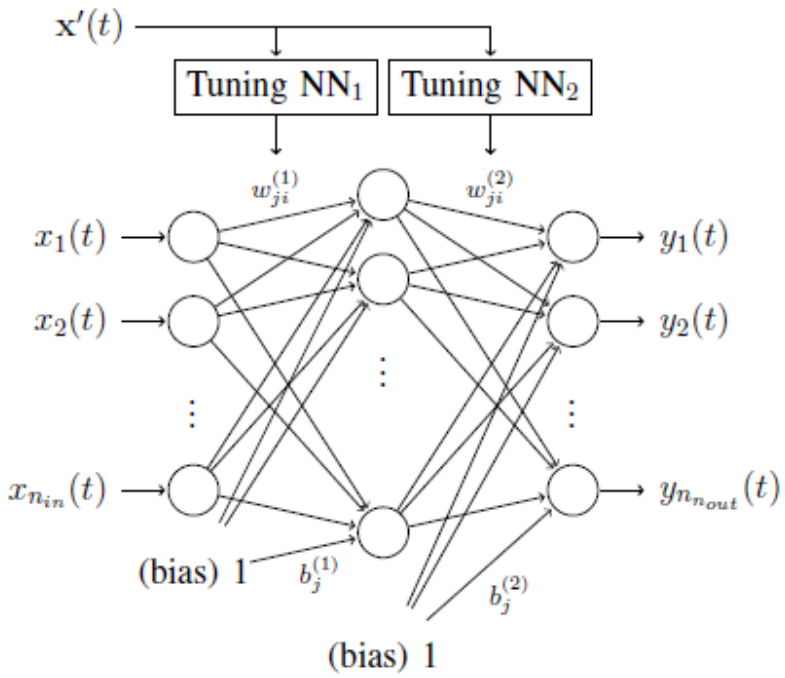


Figure 5.24: A three-layer variable-weight neural network

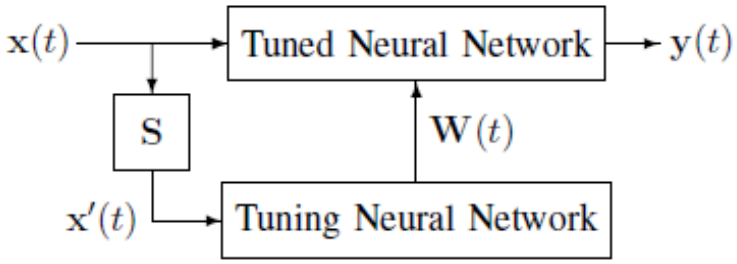


Figure 5.25: A block diagram of variable-weight neural network

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 6

Towards dynamically stable reservoirs

6.1 A specific implementation

6.1.1 Overview of the topology

We let $\mathbb{T} := [1, N_T]$ be a discrete time set and define:

- $u \in \mathbb{R}^{N_T \times N_U}$ the input signal at time t and we define $U := [1, N_U]$.
- $X := (x^k)_{k \in N_K} \in \mathbb{R}^{N_K \times N_T \times N_{X_K}}$ a sequence of reservoir neuron activations (often called kernels). $\forall k \in \mathbb{K} := [1, N_K]$, x_k is a factor which can be seen as a fraction of the reservoir. Factors are usually defined with the same matrix design and can be correlated or not (in terms of dimension and jump). By allowing this third dimension, we can represent several high dimensional signal in a same reservoir and in a dependent or independent manner. They all have their own set of Equations (5.3.8) (5.3.9) and parameters coming with it. We also define $x = [x^1 \| \dots \| x^K]$.
- $y, \ \hat{y} \in \mathbb{R}^{N_T \times N_Y}$ are respectively the output and forecast signals. And we also define $\mathbb{Y} \equiv \hat{\mathbb{Y}} := [1, N_Y]$.
- $\forall k \in N_K$, $W_{in}^k \in \mathbb{R}^{N_{X_k} \times N_U}$, $W_{res}^k \in \mathbb{R}^{N_{X_k} \times N_{X_k}}$ and $W_{fb}^k \in \mathbb{R}^{N_{X_k} \times N_Y}$ are matrices mapping respectively u to x^k, x^k to x^k and y (or $\hat{y}$) to x^k. These matrices are randomly constructed but supported by a well chosen topology. We present our choice of topology in Appendix (6.1.3). They stand for the part of the topology learned in an unsupervised manner.
- The output matrix is slightly different as we gather all the reservoir factors in a unique vector $s = [1 \| u \| x^1 \| \dots \| x^{N_K}] \in \mathbb{R}^{N_T \times N_S}$ where $N_S = 1 + N_u + \sum_{k=1}^{K} N_{X_k}$. Then, the output matrix is $W_{out} \in \mathbb{R}^{N_Y \times N_S}$. This matrix is the only part of the topology learned in a supervised manner.

Electronic copy available at: https://ssrn.com/abstract=3307566

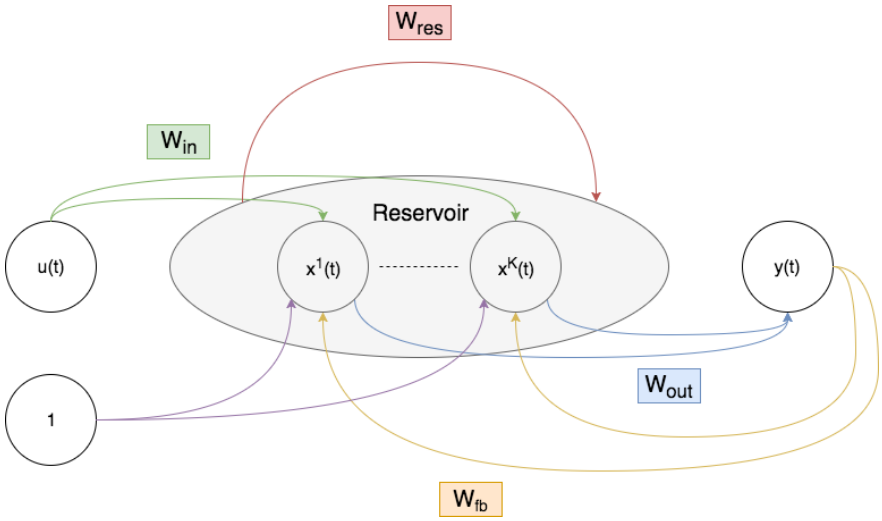


Figure 6.1: An Echo State Network with $N_x = K$ factors

6.1.2 Cost function

We are interested in the following temporal task: to find a model mapping u to y such that it minimises a given error measure. Mathematically, we can define a function ϕ corresponding to the network in Figure (6.1) as follows:

$$\begin{cases} \phi : T \times U \to Y \\ \phi : t, u \to \hat{y} = g(W_{out}[1 \| u(t) \| x(t)]) \\ x : T \times U \times X \times Y \to Y \\ x : t, u, x, y \to x(t) = f(W_{in}u(t) + W_{res}x(t-1) + W_{fb}y(t-1)) \end{cases} \tag{6.1.1}$$

Note, x behaves as a simple network but we will introduce a more complex function later.
We recall the definition of the Root-Mean-Square Error (RMSE) between the train and target vectors:

$$E(y_{train}, y_{target}) = \sqrt{\frac{1}{T \times N_y} \sum_{t=1}^{T} \sum_{i=1}^{N_y} (y_{target,i}(t) - y_{train,i}(t))^2} \tag{6.1.2}$$

6.1.3 Random generation of the input and feedback matrices

These matrices are chosen randomly as suggested by the ESN framework. They can be initialised by using a particular random variable distribution, such as, the uniform or Gaussian distribution. In our experiments we choose to use a Gaussian distribution. We also consider a scaling parameter for each matrix in order to optimise them, denoted by $s_{in}, s_{bias}, s_{res}$ and s_{fb}. As discussed in Section (1.2), the feedback scaling parameter is of key importance in our network as the reservoir needs to recall its last forecasting value in the best manner regarding our cost function defined in Equation (6.1.2).

6.1.4 Topology of the reservoir matrix

Using a completely random network leads the reservoir structure to be poorly understood. Rodan et al. [2012] demonstrated that simple cycle reservoir (SCR) shows comparable performance to the randomised one. Further, they

Electronic copy available at: https://ssrn.com/abstract=3307566

have introduced a new model adding regular jumps to the SCR, called cycle reservoir with jumps (CRJ). The latter has shown superior performance compared to the traditional ones in non-linear system identification, more particularly in time series prediction. We consider this last model in our system. For example, if each of our reservoir factor was made of six neurons, the reservoir would be as in Figure (6.2).

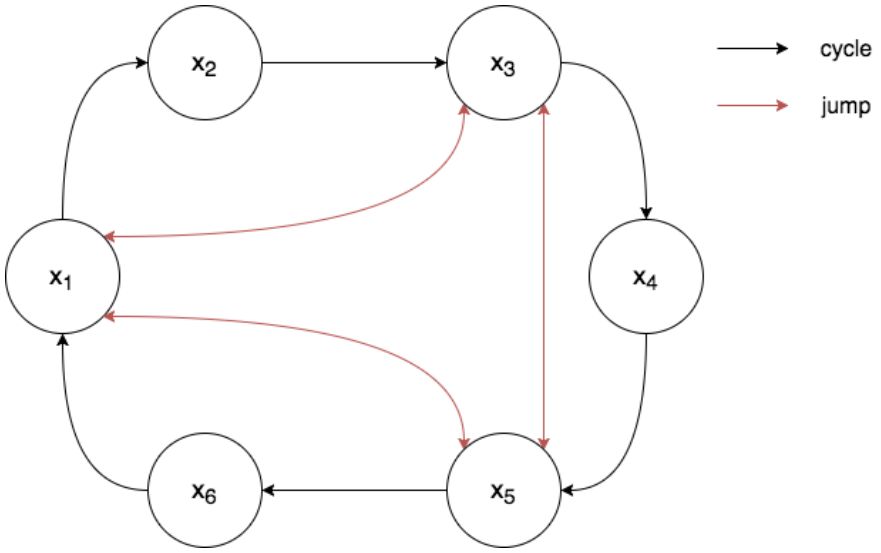


Figure 6.2: CRJ model for a 6 neurons reservoir factor

In the CRJ model, four parameters are considered for each factor: the cycle weight r_c, the jump weight r_j, the jump level telling at which neuron the jumps start and the jump sizing how many neurons are jumped. Given these parameters, the reservoir is entirely deterministic. Considering Figure (6.2) above, the reservoir matrix is constructed as follows:

$$W_{res} = \begin{bmatrix} 0 & 0 & r_j & 0 & r_j & r_c \\ r_c & 0 & 0 & 0 & 0 & 0 \\ r_j & r_c & 0 & 0 & r_j & 0 \\ 0 & 0 & r_c & 0 & 0 & 0 \\ r_j & 0 & r_j & r_c & 0 & 0 \\ 0 & 0 & 0 & 0 & r_c & 0 \end{bmatrix}$$

6.1.5 Reservoir dynamics

In a temporal task, it is important to constantly update the kernels with a function taking both new input and the previous kernels. This leads to the notion of dynamical short-term memory. The state vector x is updated at every time $t \in \mathbb{T}$ using the following recursion equations:

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\forall k \in \mathbb{K} \quad \begin{cases} \tilde{x}^k(t) = \delta^k S_{a^k}(W_{in}^k \cdot [1; u(t)] + W_{res}^k \cdot x^k(t-1) + W_{fb}^k \cdot y(t-1) + \epsilon^k) \\ \qquad\qquad + (1-\delta^k)(W_{in}^k \cdot [1; u(t)] + W_{res}^k \cdot x^k(t-1) + W_{fb}^k \cdot y(t-1) + \epsilon^k) \\ x^k(t) = (1-\gamma^k)x^k(t-1) + \gamma^k \tilde{x}^k(t) \end{cases} \tag{6.1.3}$$

where γ^k is the learning rate of the k-th kernel telling how much it learns from what has happened in the previous state as new data come in. ϵ^k is a matrix noising the kth kernel and thus preventing overfitting. It is monitored by a parameter determining the standard deviation of the Gaussian distribution. a^k is the active rate which is linked to the parametrised sigmoid function S_{a^k}. This function is applied element wise to our multidimensional factors. A plot is given in Figure (6.3), illustrating its behaviour as a varies.

$$S_a : x \in \mathbb{R} \to \frac{2}{1 + e^{-\frac{x}{a}}} - 1 \tag{6.1.4}$$

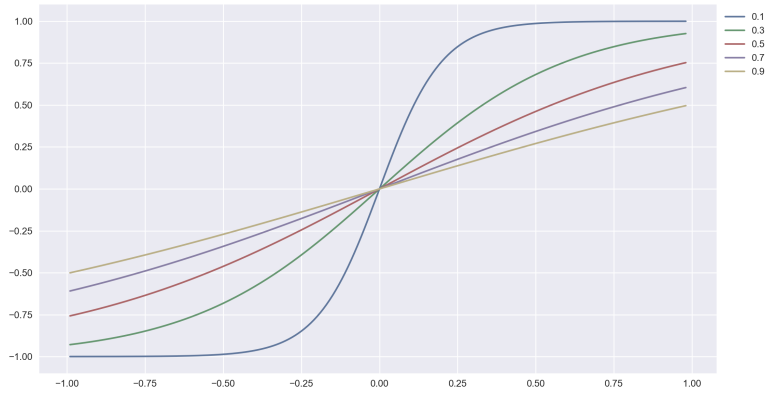


Figure 6.3: parametric sigmoid function

Sigmoidal functions are often used in machine learning to squash values to their limits. They are widely used in classification tasks. The reason being that this function is a way of assigning the negative value -1 or positive value $+1$ to a neuron in a smooth manner. The active rate a defines how important this squashing is. Further, δ^k monitors the importance of this squashing given the neuron's value.

For a given time t, the resulting $x(t)$ is called echoes of its input history $\{u(1) \dots u(t)\}$. We can set $\gamma = \alpha \times \Delta t$, where Δt is the time interval at which the data is coming into the system and α is a parameter. The smaller α and Δt are, the more the reservoir reacts slowly to new input. Note that from a signal processing point of view, the exponential moving average in Equations (6.1.3) acts as a low-pass filtering of its activations with cutoff frequency:

$$f_c = \frac{\alpha}{2\pi(1-\alpha)\Delta t}$$

6.2 Optimisation methods

We sum up the parameter set in the following picture and table. For each parameter, we provide its location, its characteristics such as its type (micro or macro) and its RMSE variability impact.

Electronic copy available at: https://ssrn.com/abstract=3307566

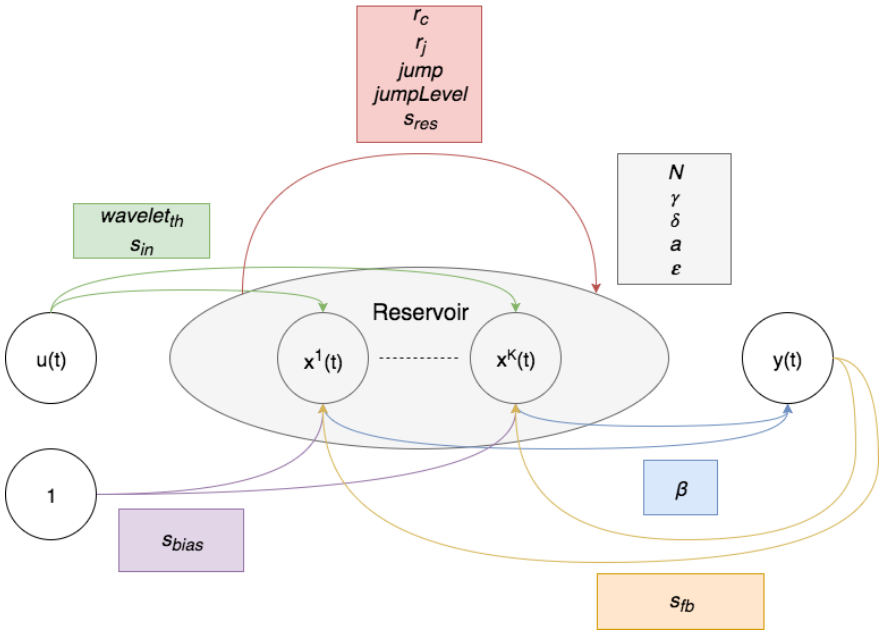


Figure 6.4: Parameters to be optimized.

	Type		RMSE Influence		
Parameter	Micro	Macro	Low	Med	High
r_c	×		×		
r_j	×		×		
$jump$	×		×		
$jumpLevel$	×		×		
$wavelet_{th}$		×			×
β	×		×		
N		×	×		
s_{in}	×			×	
s_{bias}	×		×		
s_{res}	×			×	
s_{fb}	×			×	
γ	×				×
δ	×		×		
a	×				×
ϵ	×		×		

Table 6.1: Parameter characteristics. Micro/Macro corresponds to a matrix ladder (a parameter is micro if it defines only its elements, macro if it defines its shape). Thanks to a graph of the RMSE through the optimisation process, we can classify the influence of each parameter.

Note that some parameters depend on the factor's number. The number of parameters can grow very fast as we increase this number. Only β and $wavelet_{th}$ are independent of the reservoir shape. If we define m_f as the factors' number,

Electronic copy available at: https://ssrn.com/abstract=3307566

then we have the following number of parameters:

$$n_{param} = 13 * m_f + 2 \tag{6.2.5}$$

We have focused on a 2-factor reservoir in our study as it provides two high dimensional signals and at the same time keeps the optimisation time relatively low. Therefore, the number of parameters is 28. This is relatively high and we cannot rely on an exhaustive line search, as it would take too much time to run, and also the system would be over-optimised, leading to overfitting the data.

6.2.1 Random Generation: GEN

One way of solving the above problem is to consider the following functions:

- global optimisation: it generates uniformly random values inside a space $globalMax$ times in a same loop and update the best parameters upon the RMSE obtained on the validation sample by running training and validation.
- local optimisation: exactly the same as the global one with $localMax$ times, except that it is now restricted to a smaller space (20% of the global one).
- cross optimisation: it consists in optimizing the parameters one by one $crossMax$ times and see if there is any improvement on the validation sample. It also performs two local optimisations at the end (one at 20% followed by one at 10%).
- While loop: it combines a more exhaustive global optimization ($whileMax$ times) with local and cross optimisations if there is improvement in the global.

***Remark* 6.2.1** *Note that at each call of any of these functions, new training and validation running are performed.*

Algorithm 9 Optimisation Strategy 1: GEN

Require: get $whileMax$.
Require: set $count \leftarrow 0$, $best_{RMSE} \leftarrow \infty$.
$RMSE \leftarrow$ global()
$RMSE \leftarrow$ local()
$RMSE \leftarrow$ cross()
while $count < whileMax$ **do**
  p $\leftarrow$ random()
  $node \leftarrow$ Reservoir(**p**)
  $prenode \leftarrow$ LeakyIntegrator($node$)
  $prenode$.train()
  $RMSE \leftarrow prenode$.valid()
  if $RMSE < best_{RMSE}$ **then**
    $RMSE \leftarrow$ cross()
    $RMSE \leftarrow$ local()
  end if
  $count \leftarrow count + 1$
end while
return $RMSE$

Electronic copy available at: https://ssrn.com/abstract=3307566

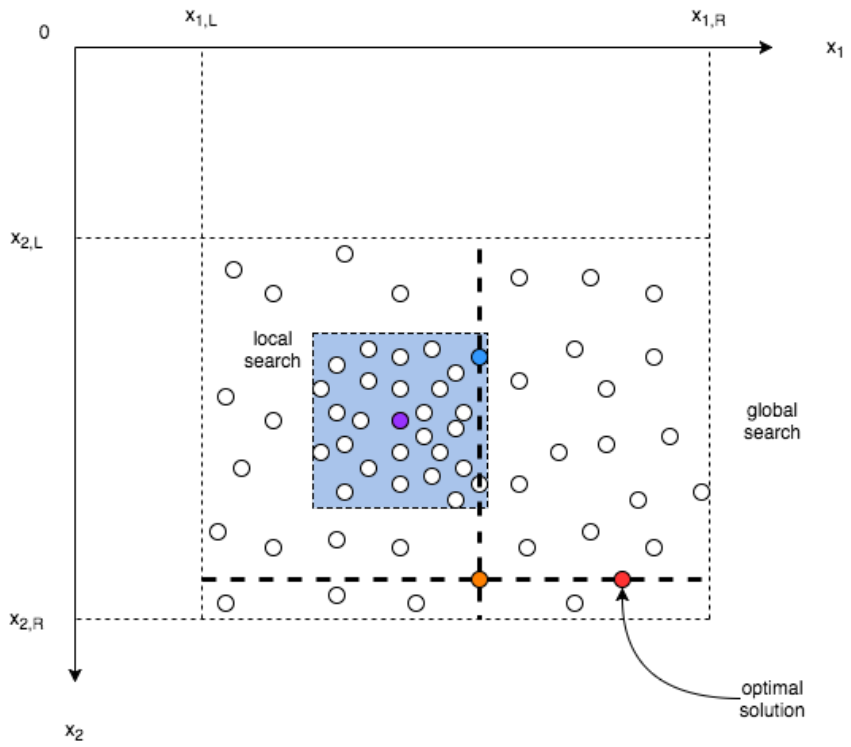


Figure 6.5: Random optimisation strategy with 2 parameters. The global optimisation finds an optimal point (purple). Then a local optimisation is done in the blue area where another optimum is found (blue). Then comes the cross optimisation where an optimum is firstly found along x_2 (orange) and secondly along x_1 (red). The $while$ loop part is not represented here in order to lighten the figure but the reader can easily see what it consists in.

***Remark* 6.2.2** *In our experiments, we note that the while loop is often useless when the cross optimisation is long enough.*

6.2.2 Stochastic Gradient Descent: SGD

The idea of this optimisation strategy is to still perform a global search and also try to find as quickly as possible the local optimum by performing a stochastic gradient descent.

Electronic copy available at: https://ssrn.com/abstract=3307566

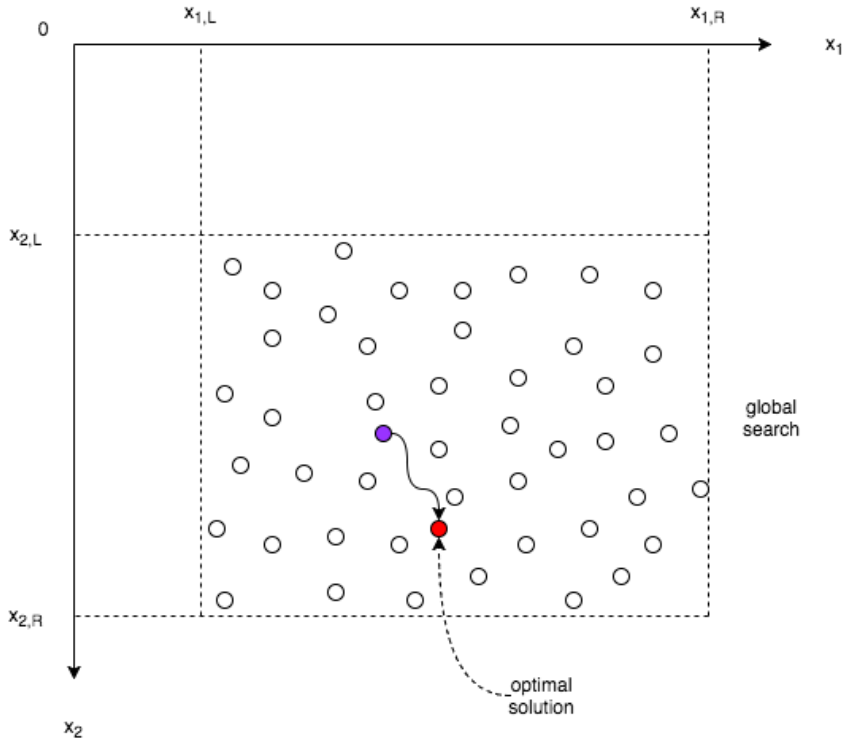


Figure 6.6: SGD optimisation strategy with 2 parameters. For every global generation (purple) we refine it by changing only a few parameters in order to get to a local optimum (red).

Tang et al. [2014] implemented this strategy by changing only the parameters r_c and r_j. They claimed that this method converges faster to an optimum and does not affect the accuracy. In this section, we lay out this method for the following set of parameters: r_c, r_j, δ, γ and a. The authors also define the reservoir matrix W_{res} with a cycle reservoir with jumps (CRJ). The idea of their method is to train the system with a hybrid optimisation strategy:

- Readouts: the output weights of the matrix W_{out} is trained using Equation (5.3.10).
- Reservoir: each parameter θ is optimised on a grid search. E is defined as in Equation (6.1.2).

$$\frac{\partial E}{\partial \theta} = 2 \sum_{t=t_0+1}^{t_1} (\hat{y}(t) - y(t)) W_{out} \frac{\partial x(t)}{\partial \theta} + \underbrace{2 \sum_{t=t_0+1}^{t_1} (\hat{y}(t) - y(t)) x(t) \frac{\partial W_{out}}{\partial \theta}}_{=0} \tag{6.2.6}$$

where θ is the parameter we optimise. Indeed, the second term is null as we use W_{out} from Equation (5.3.10), which implies that $\frac{\partial E}{\partial \theta} = \sum_{t=t_0+1}^{t_1} (\hat{y}(t) - y(t)) x(t) = 0$.

***Remark* 6.2.3** *Note that this fact is only true when W_{out} is constant, which is not the case in online algorithms. Thus, we will mainly use the GEN optimiser in our study in order to fairly compare the different methods.*

Recall the two equations updating a factor of the reservoir at time t:

$$\begin{cases} \tilde{x}(t) = \delta S_a(W_{in} \cdot [1; u(t)] + W_{res} \cdot x(t-1) + W_{fb} \cdot y(t-1) + \epsilon) \\ \qquad\quad + (1-\delta)(W_{in} \cdot [1; u(t)] + W_{res} \cdot x(t-1) + W_{fb} \cdot y(t-1) + \epsilon) \\ x(t) = (1-\gamma)x(t-1) + \gamma \tilde{x}(t) \end{cases} \tag{6.2.7}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

From these equations we can derive $\frac{\partial x(t)}{\partial \theta}$:

$$\begin{cases} \frac{\partial x(t)}{\partial r_c} = (1-\gamma)\frac{\partial x(t-1)}{\partial r_c} + \gamma(1-\delta+\frac{\delta}{a}S_a(\omega)(1-S_a(\omega)))\odot(\frac{\partial W}{\partial r_c}x(t-1)+W\frac{\partial x(t-1)}{\partial r_c}) \\ \frac{\partial x(t)}{\partial r_j} = (1-\gamma)\frac{\partial x(t-1)}{\partial r_j} + \gamma(1-\delta+\frac{\delta}{a}S_a(\omega)(1-S_a(\omega)))\odot(\frac{\partial W}{\partial r_j}x(t-1)+W\frac{\partial x(t-1)}{\partial r_j}) \\ \frac{\partial x(t)}{\partial \delta} = (1-\gamma)\frac{\partial x(t-1)}{\partial \delta} + \gamma(S_a(\omega)-\omega) \\ \frac{\partial x(t)}{\partial \gamma} = -x(t-1)+\tilde{x}(t) \\ \frac{\partial x(t)}{\partial a} = (1-\gamma)\frac{\partial x(t-1)}{\partial a} - \gamma\delta\frac{u}{a^2}S_a(\omega)(1-S_a(\omega)) \end{cases} \tag{6.2.8}$$

where $\omega = W_{in}\cdot[1;u(t)] + W_{res}\cdot x(t-1) + W_{fb}\cdot y(t-1)+\epsilon$. Then we update parameter θ using the following equation:

$$\boxed{\theta_{n+1} = \theta_n - \alpha_\theta * \frac{\partial E}{\partial \theta_n}} \tag{6.2.9}$$

Algorithm 10 Optimisation Strategy 2: SGD

Require: get $whileMax$, SGD_{Max}, V (parameters' set).
Require: set $RMSE$, count $\leftarrow 0$, $whileCount \leftarrow 0$, $SGD_{count} \leftarrow 0$ **p**.

```
while whileCount < whileMax do
    p ← random()
    while SGD_count < SGD_Max do
        for θ in V do
            θ = θ − α_θ * ∂E/∂θ
        end for
        node ← Reservoir(p)
        prenode ← LeakyIntegrator(node)
        prenode.train()
        RMSE ← prenode.valid()
        SGD_count ← SGD_count + 1
    end while
end while
return RMSE
```

6.3 Adaptive filters for online learning

There exists several ways of defining an online training algorithm, such as gradient descent and recursive least squares (see Haykin [2000], Bishop [2006]).

6.3.1 A review of adaptive filtering theory

We consider the adaptive filtering theory and justify its ability to improve the offline model presented in the previous part. Douglas [1999] gave the following definition:

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 6.3.1 *Adaptive Filter*
An adaptive filter is a computational device that attempts to model the relationship between signals in real time in an iterative manner.

Later, Steil [2004] argued that an adaptive filter is able to adapt extremely fast to the main characteristics of the task in a sort of one-shot online-learning.
In general, an adaptive filter can be identified by four different features:

1. The signal being processed by the filter. For instance, it can correspond to the daily closed prices.
2. The structure/scheme/process that is used to compute the output signal from the input signal it processes. This is represented by Figure (6.1).
3. The parameters defined in its structure/scheme/process that can be modified in an iterative manner. The aim being to modify the input-output relationship to reduce a cost function.
4. The algorithm describing how these parameters are modified through time.

Given $\mathbb{T} = [1, N_T]$ a discrete time set, for each $t \in \mathbb{T}$, the input signal $u(t)$ is fed into the reservoir, now called adaptive filter, which computes an output signal $\hat{y}(t)$. The reservoir defines parameters that can adjust the values of $\hat{y}(t)$. At time $t+1$ we know the real world value of the output, therefore it is natural to define the difference signal as follows:

$$\forall t \in [1, T], \ e(t) = y(t) - \hat{y}(t)$$

This signal, described in the diagram in Figure (6.7), is called the error signal. As time t is incremented, we hope that our adaptive filter learns from the environment and gets better and better at matching the response signal $y(t)$ through an adaptation process.

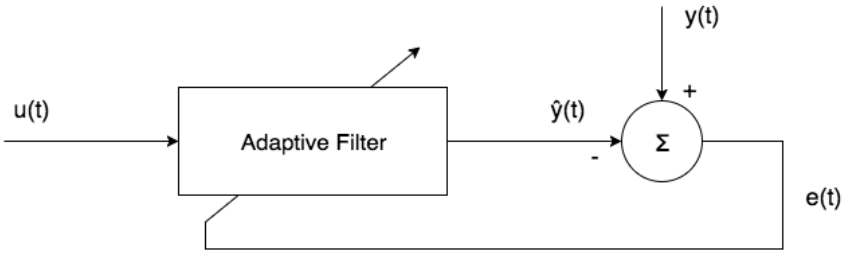


Figure 6.7: Scheme of the adaptive filtering purpose

There exists several types of application such as system identification, inverse modelling and linear prediction. For our task, we will use the latter and describe it in the diagram in Figure (6.8).

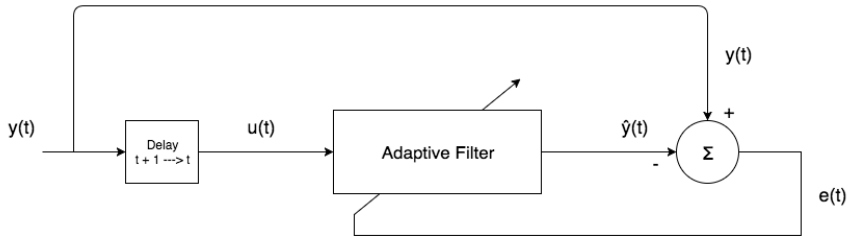


Figure 6.8: Linear Predictor problem

Electronic copy available at: https://ssrn.com/abstract=3307566

In our task, we have the following:

$$\forall t \in [1, T]\,,\ u(t) = y(t - \Delta)$$

with $\Delta = 1$ when the non-filtered data are fed in the adaptive filter.
The diagram above indicates that if a certain output signal $y(t)$ is desired, we can consider it as input to compute the output of our model $\hat{y}(t)$. If the system we wish to predict is stable enough, then the next forecast should be even closer.
There are several measures to quantify the task performance such as the root mean square error (RMSE). An adaptive algorithm aims at adjusting a set of parameters of the corresponding adaptive filter in order to reduce as much as possible the RMSE. Given Definition (6.3.1), an adaptive signal must define some parameters which can be adapted at every point in time to achieve this goal. From RC theory, since the significant matrix in the learning process is W_{out}, we should only adapt the latter. A general equation to model an adaptive filtering algorithm in our task is as follow:

$$W_{out}(t+1) = W_{out}(t) + \mu(t) G\left(e(t), u(t), s(t)\right)$$

where $G(\cdot)$ is a vector-valued non-linear function, $\mu(t)$ a parameter controlling the step size at time t. $e(t)$ and $u(t)$ are respectively the error and input signals evaluated at time t. Douglas [1999] also suggested *"$s(t)$ [as] a vector of states that store pertinent information about the characteristics of the input and error signals and/or the coefficients at previous time instants."* s stands as the reservoir states in our task. The author pointed out that the success or failure of an adaptive filtering algorithm heavily relies on the choice of the step size μ. Therefore, μ requires significant attention when optimising in the validation set.

6.3.1.1 Simple gradient descent

We recall the gradient descent update equation below:

$$\boxed{\omega_{i,j}(t+1) = \omega_{i,j}(t) - \mu_t \cdot \frac{\partial SE(t)}{\partial \omega_{i,j}(t)}} \tag{6.3.10}$$

where $\omega_{i,j}(t)$ is the (i,j)th element of the output matrix W_{out} at time t. We also recall the equation computing the forecast at time t:

$$\hat{y}(t) = g\left(W_{out} s(t)\right) \tag{6.3.11}$$

where $g \in C^1(\mathbb{R})$ and applies to a vector in an element-wise manner and $\forall t \in \mathbb{T}\ \ s(t) = [1 \| u(t) \| x(t)]$.
We derive $\frac{\partial E(t)}{\partial \omega_{i,j}(t)}$ below using the framework defined in Appendix (4.1.3).

$$SE(t) = \sum_{i=1}^{N} \left(y_i(t) - \hat{y}_i(t)\right)^2 \tag{6.3.12}$$

$$\Rightarrow \quad \frac{\partial SE(t)}{\partial \omega_{i,j}(t)} = -2 \sum_{i=1}^{N} \left(y_i(t) - \hat{y}_i(t)\right) \frac{\partial \hat{y}_i(t)}{\partial \omega_{i,j}(t)} \tag{6.3.13}$$

$$\Rightarrow \quad \frac{\partial SE(t)}{\partial \omega_{i,j}(t)} = -2 \sum_{i=1}^{N} \left(y_i(t) - \hat{y}_i(t)\right) g'\left(W_{out} s(t)\right) s_j(t) \tag{6.3.14}$$

6.3.1.2 Recursive least squares

The Recursive Least Squares (RLS) algorithm can be used for iterative computation of the inverse of the correlation matrix R_{ij}^{-1} and the readout weights ω (see Appendix (5.3.1)). It can be summarised as follows:

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}\kappa(t) &= \frac{\nu^{-1}\Gamma(t-1)x(t)}{1+\nu^{-1}x^{\top}(t)\Gamma(t-1)x(t)}\\ \omega(t) &= \omega(t-1)+\kappa(t)\left(y(t)-\hat{y}(t)\right)\\ \Gamma(t) &= \nu^{-1}\Gamma(t-1)-\nu^{-1}\kappa(t)x^{\top}(t)\Gamma(t-1)\end{aligned}$$

where $\kappa(t)$ is the gain vector and $\Gamma(t)$ is the estimate of the inverse of the correlation matrix. The forgetting factor ν is set to 1. This method ensures faster convergence of the readout weights but is much more computationally intensive than the simple gradient descent algorithm.

6.3.2 Online algorithm

We are now going to describe the online algorithm for real time learning of output matrix W_{out}. While we have considered the actual output of the financial time series y, it is more relevant to take the filtered signal as the target signal. Indeed, this signal is by definition smoother and therefore more continuous in time.

Algorithm 11 RC online Training

Require: choose a gradient descent algorithm $\mathbb{A}$

1: $W_{out} \leftarrow$ Algorithm (8)
2: set $SE \leftarrow 0$
3: **for** $t \leftarrow 0$ **to** T_{valid} - 1 **do**
4: update each reservoir factor using Equation (6.1.3).
5: compute forecast using Equation (6.3.11).
6: $SE \leftarrow SE + (y(t+1) - \hat{y}(t+1))^2$
7: update W_{out} with Equation (6.3.10) using $\mathbb{A}$.
8: **end for**

In Algorithm (11) we use $\mathbb{A}$ as the scaled conjugate gradient implemented in GNU Scientific Library [2017]. This method is very fast for a well-chosen initial guess and it makes sense to suppose that the output matrix should be close to its previous state as the signal we try to calibrate is smooth.

***Remark* 6.3.1** *We can also replace y in Equation (6.1.3) with its last feedback $\hat{y}$. Consequently, the reservoir has more information about how accurate its last forecast was.*

6.4 Some results

In order to gain some insight on the capability of neural networks to reproduce some dynamics, we will test the Reservoir Computing (RC), in particular the implementation given in Section (6.1), on the Mackey-Glass signal and some FX time series. The model will be tested both with offline and online algorithms.
We reproduce below some results on the Mackey-Glass signal and FX series obtained by a student of ours who did his MSc thesis at QFL (see Bouizi [2017]).

6.4.1 Using Offline algorithms

6.4.1.1 The Mackey-Glass series

In the academic literature, it is found that pattern generation can be easily implemented and produces very good results. However, the Mackey-Glass series prediction task is more complex and requires a large reservoir and more elaborate

Electronic copy available at: https://ssrn.com/abstract=3307566

training algorithm (see Wyffels et al. [2009], Antonik et al. [2016]).

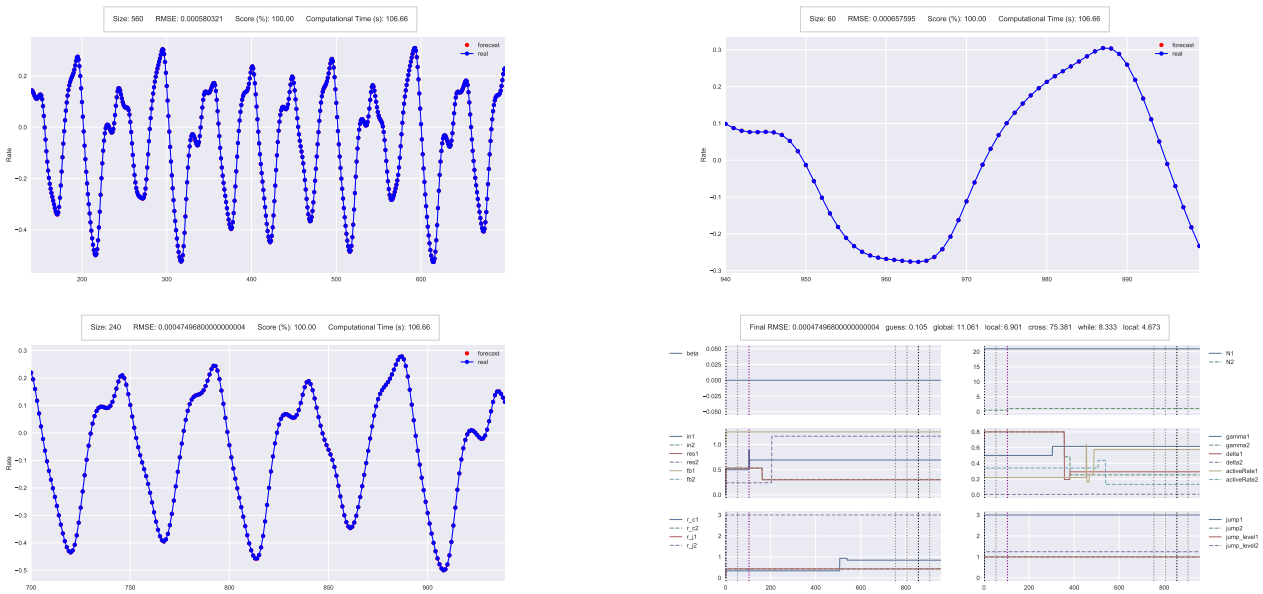


Figure 6.9: Mackey-Glass time series forecasting. On the left hand side from top to bottom: Training and Validation samples. On the right hand side from top to bottom: Testing and parameters optimisation through the GEN optimisation process. A black dotted vertical line indicates the beginning of a global or $while$ loop optimisation. A gray one corresponds to a local one and a purple one indicates a cross optimisation.

Method	RMSE	Ranking
Neural tree (see Chen et al. [2004])	0.0069	5
PNN (see Tan [1995])	0.0059	4
NN + Wavelet Denoising (see Tan [2009])	0.0028	3
NN + Wavelet Packet Denoising (see Tan [2009])	0.0024	2
ESN offline	0.00065	1

Table 6.2: Comparison of our results with other research papers on the testing sample.

From Table (6.2) above, we can conclude that our method is very accurate at forecasting the Mackey-Glass time series and clearly outperforms classical methods. It is important to note that GEN optimisation method (see Appendix (6.1.3)) has reduced the RMSE from 0.00545518 to 0.000474968 on the validation sample. One of the reason for getting better results is that we optimise a large number of parameters in a proper manner without overfitting the data and in a relatively short period of time (approximately 00:01:36).

***Remark* 6.4.1** *It is not surprising to get good accuracy as this signal exhibits clear information without any noise. In this experiment, we only assess the ability of our neural network to incorporate its behaviour and to reproduce it step by step as redundant information comes in.*

Electronic copy available at: https://ssrn.com/abstract=3307566

6.4.1.2 Financial time series

Following the same approach on financial time series, we reproduce some results on the FX markets. We perform a GEN optimisation (see Section (6.1.3)) with offline training using the following parameters: $(globalMax, crossMax, localMax, whileMax) = (20, 20, 20, 0)$. Considering Remark (6.2.2), we choose the following setting $whileMax = 0$. We study the specific FX pair: EURGBP from 2010-01-01 to 2017-05-01. From Figure (6.10) below, we observe good results on the training and validation samples with a relatively low RMSE and a good score (almost 60%). However, Figure (6.10) suggests that testing is not satisfying, with a high RMSE at 0.0440 and a score below 50%.

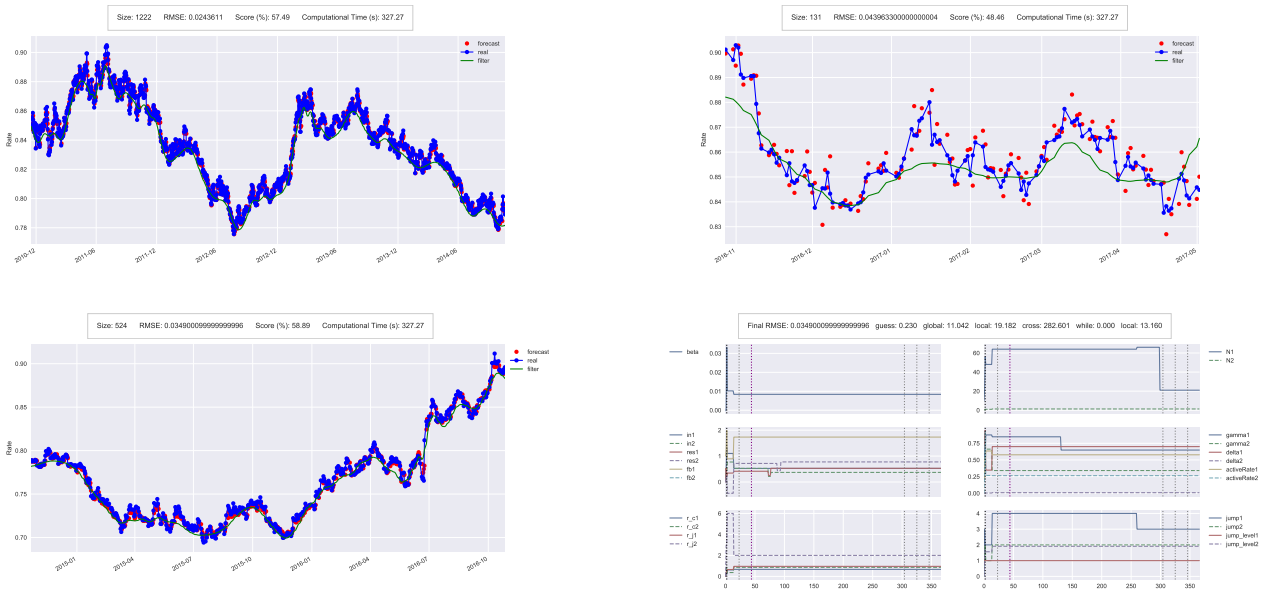


Figure 6.10: EURGBP time series forecasting. On the left hand side from top to bottom: Training and Validation samples. The filter (green curve) is used as input on these samples. On the right hand side from top to bottom: Testing and parameters optimisation through the GEN optimisation process. It is important to precise that the filter (green curve) is not used during testing and therefore is purely informative.

So far, we have worked with a single FX pair. In order to perform a broader study, we consider a pool of instruments from the same market and focus on two cases: one with filtered data and one with non-filtered data. We have gathered our results and computed the mean of the RMSE and Score for each sample.

	Filtered Input			Non-Filtered Input		
	Training	Validation	Testing	Training	Validation	Testing
RMSE	0.0199	0.0282	0.0290	0.0241	0.0336	0.0237
Score	0.6207	0.6692	0.5045	0.4860	0.5196	0.5126

Table 6.3: Comparison between filtered input and non-filtered input results on 71 FX pairs.

From Table (6.3), one can clearly observe that we get better learning when using filtered input data. However, we can not use a denoising technique when testing the data as it would result in data-snooping bias. Thus, we are left with using market data, which results in poor performance (slightly above 50%). Intuitively, we have trained a reservoir with filtered data. As long as the new input data is relatively smooth, the reservoir can produce good predictions.

Electronic copy available at: https://ssrn.com/abstract=3307566

However, in presence of noise the reservoir state is too much disrupted and gets too far away from a stable state. When using non-filtered data, the table shows that there is no learning. It corresponds to case (3) above where the network is trained with an attractor-based model but exploitation is ran on a transient process.

6.4.2 Using online algorithms

In the case of the Mackey-Glass time series discussed in Section (6.4.1), we have shown that the offline algorithm was very good at calibrating and predicting this signal. However, we have seen that the performance was downgraded when adding a noisy component to it (equivalent to financial time series).
Both academics and practitioners proposed to smooth financial data with filtering methods. For instance, wavelets were considered as they can represent any signal in a smooth manner and, as such, provide a certain stability (see Appendix (14.4)). Further, it was suggested that they give information about the past and future at any time t in the training and validation samples, teaching the model how to behave. A good overview of the application of wavelets in economics and finance is given by Ramsey [1999].

6.4.2.1 The Mackey-Glass series

For simplicity of exposition, we consider the following framework:

$$\forall t \in \mathbb{T} \qquad f(t) = f^*(t) + \epsilon(t) \tag{6.4.15}$$

where ϵ is a Gaussian white noise with an appropriate constant width. We also suppose that there exists a wavelet $\hat{f^*}$ capable of approximating f^*. We want to show that learning with an online algorithm produces better approximation results than with an offline algorithm. We consider the Mackey-Glass signal presented in Section (1.3.0.1). In the training and validation samples, f^* is used as target y in the Algorithm (11). Again, we cannot use it in the testing sample as it would result in data-snooping bias. Therefore, the green signal is only informative. The signal which is used in the testing sample by Algorithm (11) is the one coloured in magenta in Figure (6.13). It is the result of Algorithm (38) applied at every time t in the testing sample (from the beginning of validation).

Electronic copy available at: https://ssrn.com/abstract=3307566

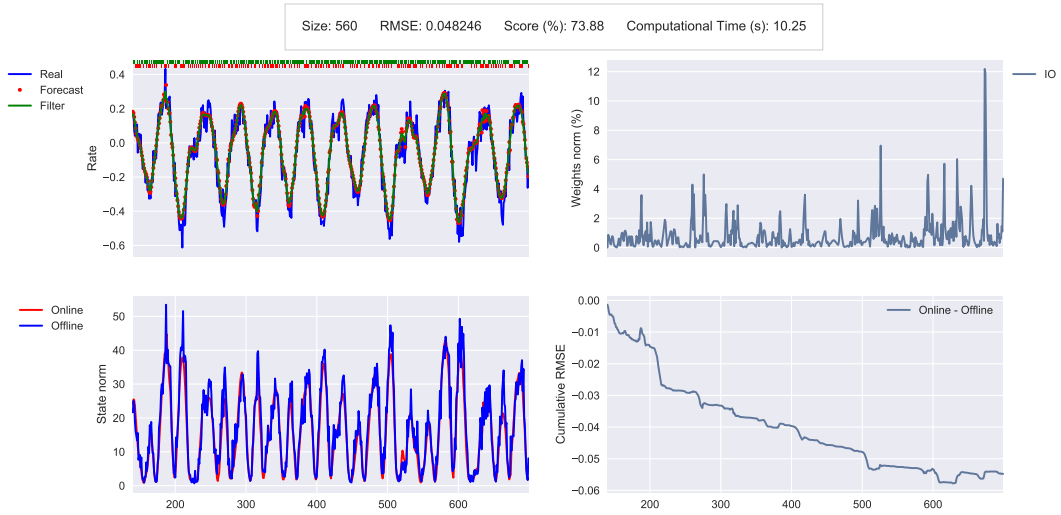


Figure 6.11: Noised Mackey-Glass online forecasting on the training sample. Top Left: Original, target filtered signals and forecasts. We have also plotted the local accuracy forecast (+1 if real slope is well forecast, -1 if not). Bottom Left: Reservoir states L2-norm from offline and online algorithms. Top Right: Normalised output weights evolution $\frac{||W_{out}(t)-W_{out}(t-1)||}{W_{out}(0)}$. where $W_{out}(0)$ is the offline matrix calibrated on the training sample. Bottom Right: error $cumulRMSE_{on} - cumulRMSE_{off}$.

From Figure (6.11) above, one can see that the denoised signal using Algorithm (38) is very smooth and has the same shape as the original Mack-Glass time series. Further, the learning Algorithm (11) leads to very good prediction of the green signal. This good learning is guaranteed by the combination of smooth states (plot on the bottom left, better visualisation on Figure (6.13)) and the adaptive filtering technique refining the output weights at each point in time (plot on the top right). This technique results in a very good approximation of the filtered signal and clearly outperforms the offline algorithm. Overall, we get a very good RMSE at 0.048246 and score at 73.88%. Note that we have used only a few iteration in this simulation: $(globalMax, crossMax, localMax) = (1, 1, 1)$. It indicates that a lot of reservoirs can be appropriate for our task, adding more flexibility. Also it is very computationally efficient as a good learning can be produced in less than 11 sec.

Electronic copy available at: https://ssrn.com/abstract=3307566

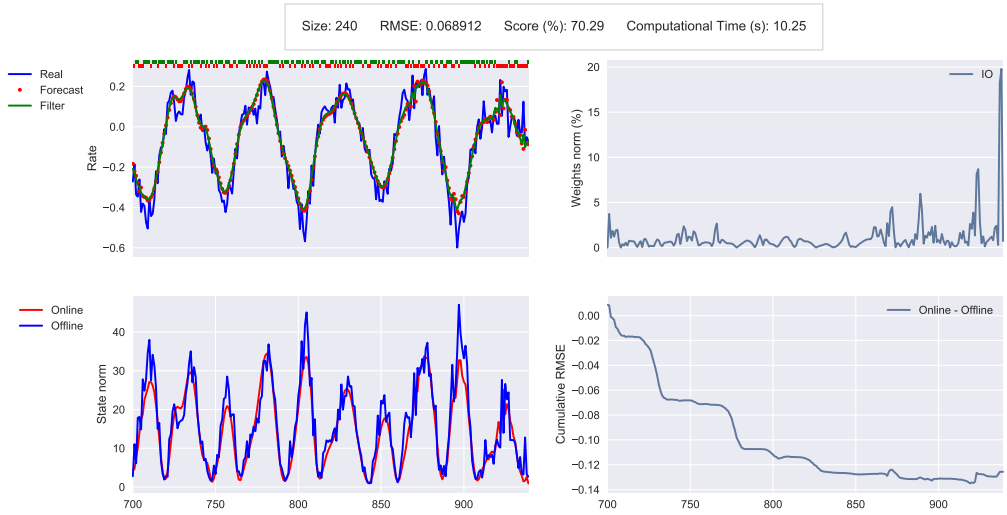


Figure 6.12: Noised Mackey-Glass online forecasting on the validation sample. See Figure (6.11) for an advanced description.

From Figure (6.12) above, we see that the good prediction performance is preserved on the validation sample and that the algorithm produces roughly the same results than the one obtained on the training sample.

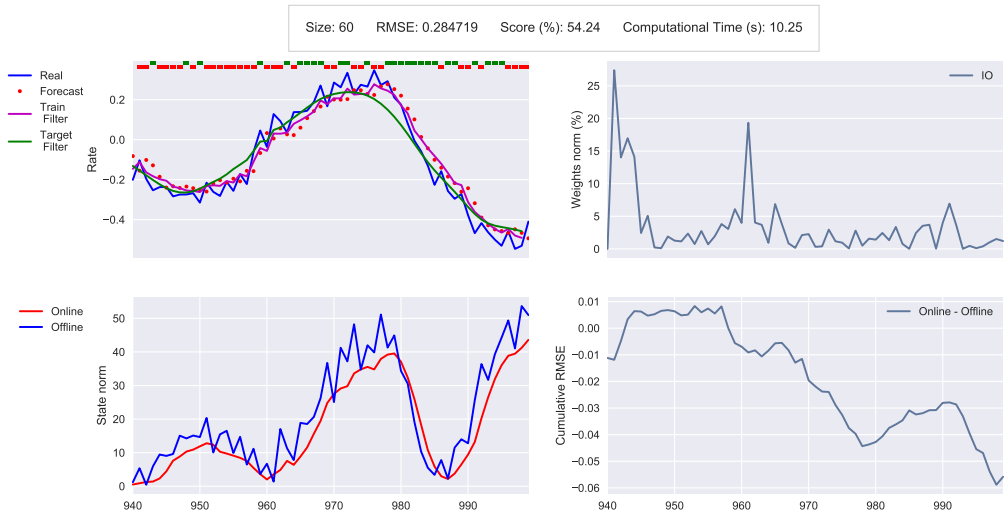


Figure 6.13: Noised Mackey-Glass online forecasting on the testing sample. See Figure (6.11) for an advanced description. We have added the filtered used by the algorithm in magenta. The green one is informative only.

Figure (6.13) shows that the magenta signal produces good results as the states remain stable. However, the results of the online algorithm against the offline one are not as good. Further, the output weights are still changing (from 5% to 25%), indicating that the online algorithm cannot forecast constant parameters in time. This is undesirable as we would like our reservoir to have good predictive properties when only feeding it with the new filtered input.

Electronic copy available at: https://ssrn.com/abstract=3307566

6.4.2.2 Financial time series

We now assess the performance of the online algorithm on market data. Again, we use our pool of 71 FX pairs from 2010-01-01 to 2017-05-01.

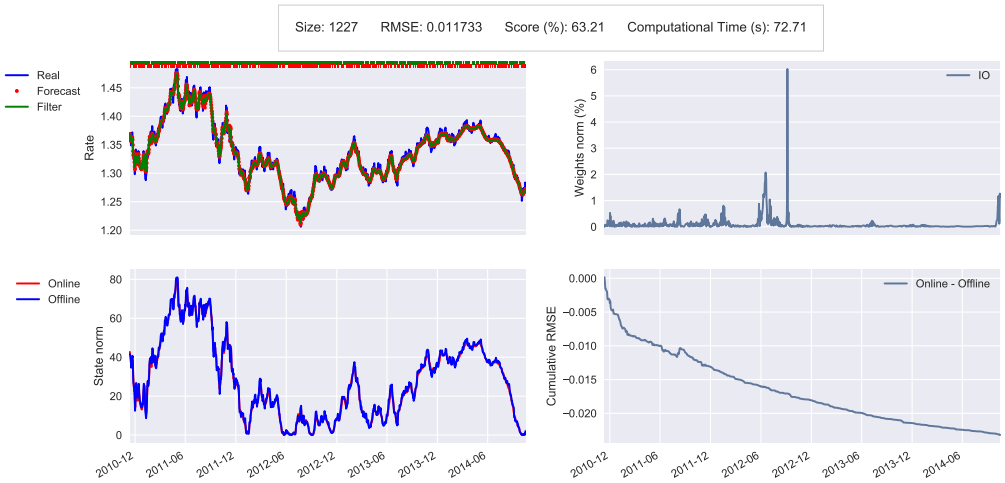


Figure 6.14: EURUSD online forecasting on the training sample.

Figure (6.14) above demonstrates good learning and good performance of the online algorithm compared to the offline one. However the output matrix is sometimes unstable. For instance, a peak of nearly 6% has occurred between 2012-06 and 2012-12, which is well above the average amplitude of the relative output matrix variation. It often happens during regime shift. Further, when we look at EURGBP on the 23^{rd} of June, 2016 (Brexit), we see that the adaptive filtering has produced a peak of 4%. Overall, there is good learning with a RMSE at 0.011733 and a Score at 63.21%. The algorithm ran in approximately one minute.

Electronic copy available at: https://ssrn.com/abstract=3307566

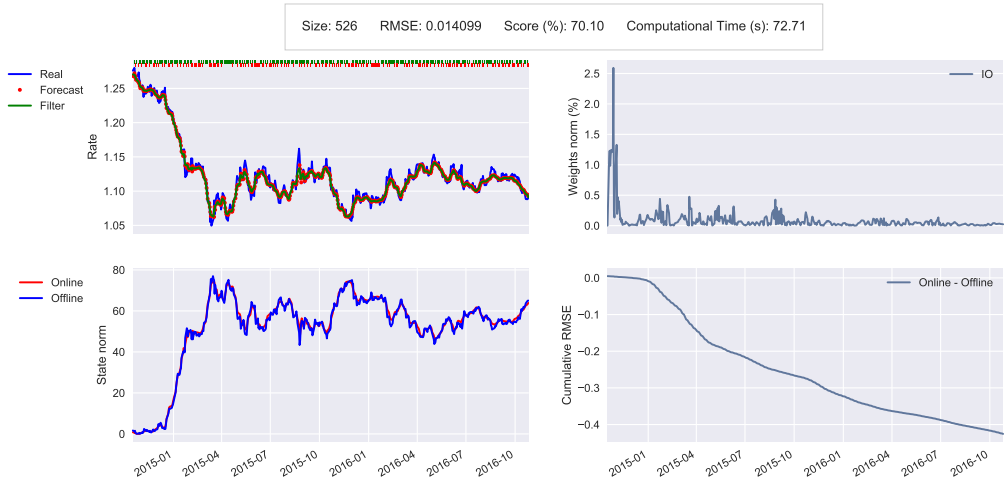


Figure 6.15: EURUSD online forecasting on the validation sample.

From Figure (6.15) above, we note that good performance is still maintained on the validation sample.

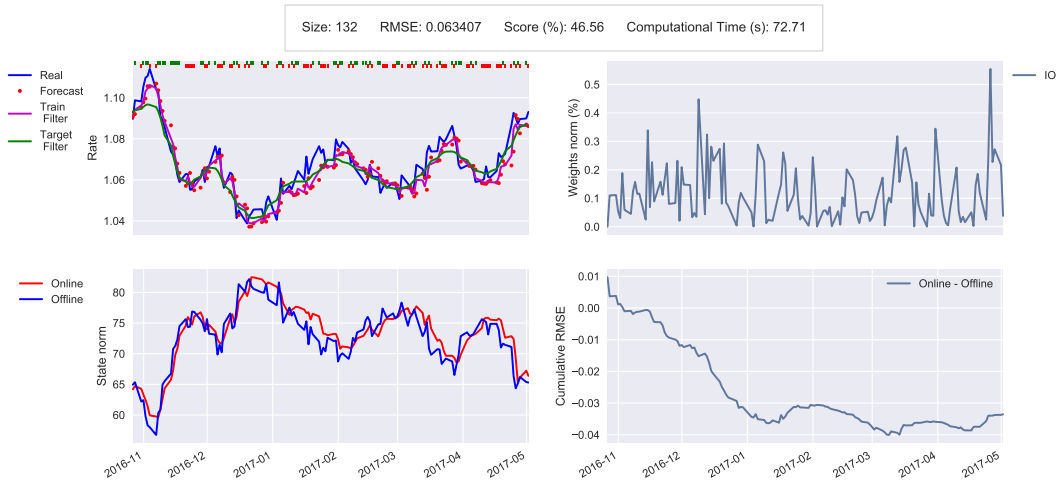


Figure 6.16: EURUSD online forecasting on the testing sample.

On the testing sample, the performance is degraded as the filter we used has no-longer information about the future. Therefore, it does not move like the green one. Nonetheless, the states norm is still smoother than the one obtained from the offline algorithm, which is desired when learning time series. Further, we observe that the output weights are varying a lot, although with small amplitude (most of the time below 0.5%). Thus, we conclude that the forecast has not reach the nearest stable point (which would correspond to the next value of the filtered series we use).

Electronic copy available at: https://ssrn.com/abstract=3307566

	Training	Validation	Testing
RMSE	0.0213	0.0314	0.0813
Score	0.6078	0.6694	0.4910

Table 6.4: Global assessment of the online algorithm on the pool of 71 FX pairs from 2015-01-01 to 2017-05-01.

In Table (6.4), we report the results on the same FX pool as the one used previously. Our results demonstrate good learning on the training and validation samples with low RMSEs and high scores. However, some issues remain on the testing sample. One could argue that remaining as close as possible to the filtered series (magenta) in the testing is crucial, while not interfering with the calibrated model by modifying some matrix (particularly the output weights).

6.5 Associative Reservoir Computing

6.5.1 Paradigm

6.5.1.1 Definitions

So far, we have taught the reservoir the input to output mapping. Equation (6.5.16) introduce the function ϕ defining this mapping and carrying the forward relation between the input sample U and the output sample Y without studying its properties. We are now going to understand the function ϕ in view of making better prediction.

Definition 6.5.1 *Ambiguous relation*
Let $\phi : U \to Y$. We say that ϕ is ambiguous if it is not injective i.e. $\exists (u_1, u_2) \in U^2 | u_1 \neq u_2 \Rightarrow \phi(u_1) = \phi(u_2)$

Definition 6.5.2 *State space*
A state space is the set of all possible states [the reservoir states x in our case] of a dynamical system. Each state of the system corresponds to a unique point in the state space (see Terman et al. [2008]).

Definition 6.5.3 *Attracting set*
Let S be a dynamical system and X its associated state space. We say of a set A to be an attracting set of S if it is a closed subset of X such that $\forall a \in A, \exists x(0) = x_0 | x(t) \to a$ as $t \to +\infty$.
We call its elements attractors.

Definition 6.5.4 *Basin of attraction*
Let a be an attractor of A. We define the basin of attraction of A $B(A)$ as the set of initial conditions such that $x(t) \to a$ as $t \to +\infty$.

In associative reservoir computing, the learning process aims at shaping the reservoir dynamics in a particular attractor. A major problem when learning ϕ is that it may be an ambiguous relation. Following Reinhart [2011], we develop a method for resolving the ambiguity of ϕ using a dynamical approach.

6.5.1.2 The model

In the ARC model, we overwrite the reservoir forward representation by a joint representation involving input u and output y. To access the latter, we need a matrix $W_{fb} \neq 0$. Furthermore, we define a matrix mapping the reservoir states to the input layer and called it $W_{rec} (\in \mathbb{R}^{N_U \times N_S})$. We want to learn the mapping ψ from output to input. Figure (6.17) defines the different connections involved in this model. We have the following system:

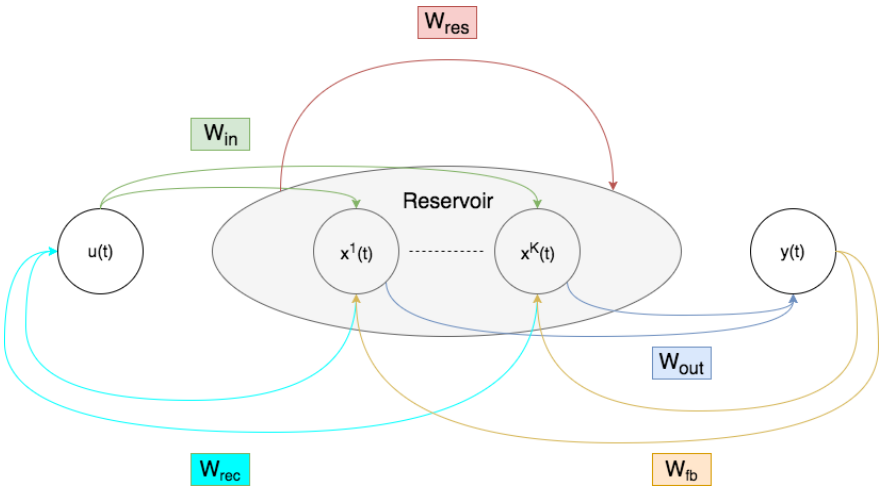
Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{cases} \phi : T \times U \times X \to Y \\ \phi : t, u, x_{fwd} \to \hat{y} = g(W_{out}[1 \| u(t) \| x_{fwd}(t)]) \\ \psi : T \times Y \times X \to U \\ \psi : t, y, x_{bwd} \to \hat{u} = g(W_{rec}[1 \| y(t) \| x_{bwd}(t)]) \\ x_{fwd} : T \times U \times X \times Y \to X \\ x_{fwd} : t, u, x, y \to x_{fwd}(t) = f(W_{in}u(t) + W_{res}x(t-1) + W_{fb}\hat{y}(t-1)) \\ x_{bwd} : T \times U \times X \times Y \to X \\ x_{bwd} : t, u, x, y \to x_{bwd}(t) = f(W_{in}\hat{u}(t-1) + W_{res}x(t-1) + W_{fb}y(t)) \end{cases} \tag{6.5.16}$$

Equations (6.5.16) above define the associative system. The system is now composed of two functions ϕ and ψ mapping respectively input to output and output to input. We have also defined two states x_{fwd} and x_{bwd} corresponding respectively to the states obtained with forward model ϕ and backward model ψ.

***Remark* 6.5.1** *In practice the backward model cannot be use in our task (at least on the testing sample) as we obviously do not know the asset price we want to predict. However this framework demonstrates that we can compute its corresponding input $\hat{u}$ by simply using the last prediction $\hat{y}(t-1)$.*

Figure 6.17: ARC model.



Typically, it is often recommended to use the output at time t when learning on the training sample, we call this process teacher forcing. On the validation set, we can keep training the W_{out} and W_{rec} matrices in order to teach the reservoir how to stay close to the target signal. That is to say, we use equation 6.3.10 to update W_{out} and W_{rec}, the idea being to stay close to an attractor as time goes on. In the next subsection we lay out our algorithm for converging to an attractor at every time.

6.5.1.3 Attractor-based computation

In the Figure (6.18) below, the system sits at time t and iterates the reservoir dynamics for a certain number of time, k_{max}, using always the same input $u(t)$. This artificial dynamics causes the reservoir $x(t)$ to evolve toward a stable state. A good forecast of the output attractor can be performed using Equations (6.5.16) afterwards.

Electronic copy available at: https://ssrn.com/abstract=3307566

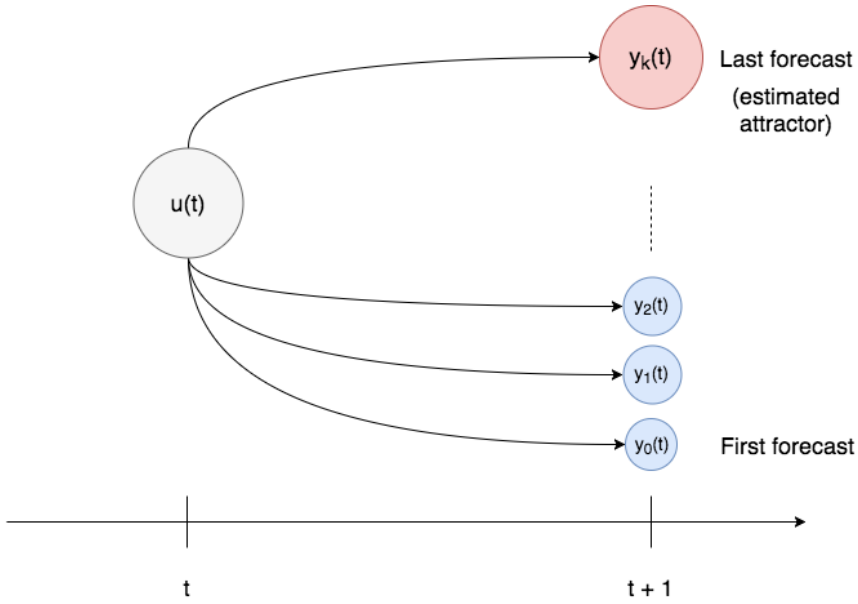


Figure 6.18: Dynamical system model to fit an attractor.

As explained by Reinhart [2011], the dynamical approach to model ambiguous inverse problems crucially depends on the ability to shape multi-stable attractor dynamics that reflect the multiplicity of solutions.

***Remark* 6.5.2** *The attractors we try to fit are the different filters we may use to denoise the initial signal. For instance, it can correspond to several wavelets, decomposition and thresholding levels.*

Algorithm 12 Attractor Convergence Algorithm

Require: get an input data at time $u(t)$.
Require: set k_{max} as a maximum number of iteration.
Require: set ϵ as a tolerance level.
1: **while** $k < k_{max}$ **and** $norm < \epsilon$ **do**
2: compute $x_{fwd}(t)$ using Equation (6.5.16).
3: compute $\hat{y}(t)$ and $\hat{u}(t)$ using Equation (6.5.16).
4: $norm \leftarrow ||x_{fwd}(k+1) - x_{fwd}(k)||^2$
5: $k \leftarrow k + 1$
6: **end while**

In Algorithm (12) we eliminate the transient effect by iterating the states using the same input. It is as if we would have frozen the series at time t and went on feeding the reservoir, leading to convergence to an attractor. Basically, we want to fit the reservoir dynamics to this attractor in order to teach the reservoir how to reproduce its behaviour. If it is the case, then the network should not need this trick any more and it would predict well this attractor. Ambiguity can be resolved by applying Algorithm (12), causing the reservoir to settle in a particular attractor.

6.5.2 States stabilisation techniques

A major issue of our task is that only noised data are observable in the market. However, the reservoir states need to be smooth (in the sense of its norm) in order to hope learning. Indeed, if the states are stables as time goes, then the

Electronic copy available at: https://ssrn.com/abstract=3307566

predicted values should be stables as well. From Table (6.3) we have seen that there is a good learning when dealing with filtered input. However we cannot use the wavelet filter in the testing sample. To offset this problem, we could imagine teaching the reservoir how to remain consistently stable while feeding it with noisy input.

6.5.2.1 Reservoir dynamics calibration to observable stable states

Suppose that we can observe stable states x^* on the training sample. We wish to find a dynamics $W_{opt} = [W_{in}\|W_{res}\|W_{fb}]$ such that it minimises the following cost function:

$$E = \sum_{t=1}^{T_{training}} ||x^*(t) - x(t)||^2 \tag{6.5.17}$$

Given u, x^* and y on the training sample, the problem is stated as follows:

$$\min_{W_{opt}} \sum_{t=1}^{T_{training}-1} ||x^*(t+1) - f(W_{opt}[u(t)\|x^*(t)\|y(t)])||^2 \tag{6.5.18}$$

Let us define

$$S^* = [[u(0)\|x^*(0)\|y(0)]\| \dots \|[u(T_{training}-1)\|x^*(T_{training}-1)\|y(T_{training}-1)]]$$

and

$$A^* = [f^{-1}(x^*(1))\| \dots \|f^{-1}(x^*(T_{training}))]$$

Then we can solve this problem using Ridge regression:

$$\hat{W}_{opt} = A^* \cdot S^{*T} \cdot (S^* \cdot S^{*T} + \lambda I)^{-1} \tag{6.5.19}$$

where λ is a well chosen regularization parameter. Given these calibrated matrices, the reservoir can now learn from noisy input while staying relatively stable.

***Remark* 6.5.3** *The filter is of key importance as it will define how the system can map noisy input to smooth states.*

In practice we want the reservoir to remain stable when fed with unseen noisy input. Thus, we must consider a large enough training sample to ensure the states remains stable for exploitation.

6.5.2.2 Learning with filtered inputs

Another idea would be to work with filtered inputs only and to directly teach the reservoir to predict the filtered data (coloured in green). Indeed, this filter can be seen as an indicator of the initial dataset. One could then argue that getting a good prediction of the filtered data could lead to good real time series forecasting if one can control the forecasting error.

Suppose that we observe stable states x^* on the training and validation samples. In these samples we can use all the information at any time, so that we can fit a filtering technique such as wavelets and feed the reservoir using the filtered inputs only. The states should then be relatively stable on the training and validation, as previously discussed. Then, using Equation (6.3.10) (for both W_{out} and W_{res}), we teach the network to fit this filtered dynamics. At the end of the validation process, we hope that our network will be well enough trained and that we no-longer need Equation (6.3.10) to obtain good forecasts. That is, we should have the functions ϕ and ψ relatively constant with respect to the parameters $(W_{in}, W_{res}, W_{fb}, W_{out}, W_{rec})$.

Electronic copy available at: https://ssrn.com/abstract=3307566

We present in Figure (6.19) our risk management scheme when making a forecast. The idea being to use, under particular assumptions, the information at time t with the backward mapping function ψ, in order to have a control over the forecast error.

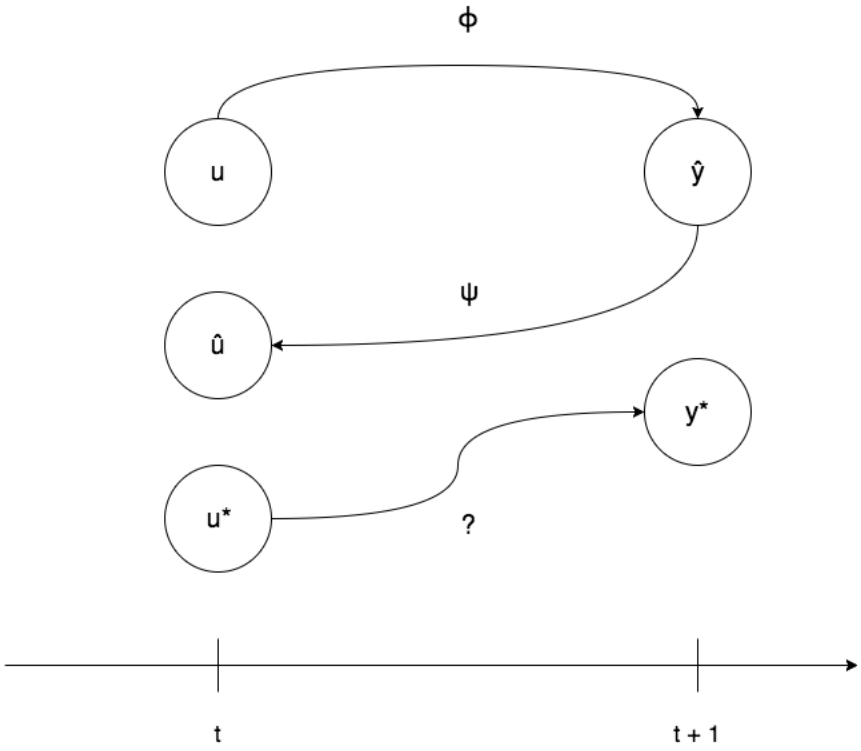


Figure 6.19: Risk management scheme.

Suppose that $\exists t_0 \in [0, T_{valid}]$ such that $\forall t \in \mathbb{T}_0 := [t_0, T_{valid}]$, $\phi(u(t)) = \hat{y}(t)$ and $\psi(\hat{y}(t)) = u(t)$. Then $\psi^{-1}_{|\mathbb{T}_0} = \phi_{|\mathbb{T}_0}$ (injective property is verified at time t if Algorithm (12) converges). Suppose also that $\phi_{|\mathbb{T}_0}$ is time-independent and that $\forall t \in \mathbb{T}_0, |u^*(t) - \hat{u}(t)| \leqslant K$. Then, we could set up the following strategy:

Algorithm 13 Out-of-sample forecasting risk management

Require: set $l_V = \sup\{|u^*(t) - \hat{u}(t)|, \forall t \in \mathbb{T}_0\}$
Require: set $l_T \leftarrow 0$
1: **while** $l_T < l_V$ **do**
2: update the reservoir x up to time t using Equations (5.3.8) and (5.3.9).
3: compute forecast $\hat{y}(t)$ and associative input $\hat{u}(t)$ using Equation (6.3.11).
4: $l_T \leftarrow |u^*(t) - \hat{u}(t)|$
5: **end while**

The ARC framework is very powerful as it allows us to evaluate the input error at time t to have an idea of the forecast accuracy at time $t + 1$.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 7

Introduction to unsupervised learning

We check the performance of the Recurrent Radial Basis Function by fitting and predicting chaotic time series. This could be verified by assessing its performance on the classic Mackey-Glass series (see Figure (5.13)). We also verify performance of the models on denoised financial time series. We let the Linear Regression (LR) model be the benchmark (see Section (8.3.1)). We reproduce below some results on the Mackey-Glass signal obtained by a student of ours who did his MSc thesis at QFL (see Wang [2017]).

7.1 Batch intrinsic plasticity

7.1.1 Description

So far there are no generalised unsupervised learning algorithms that are powerful enough to be able to fully train ESN reservoirs, although several attempts were made (see Lukosevicius [2012b]). However, sometimes pre-training methods in simple machine learning models could be enlightening and useful as well in complex models such as ESN.

Electronic copy available at: https://ssrn.com/abstract=3307566

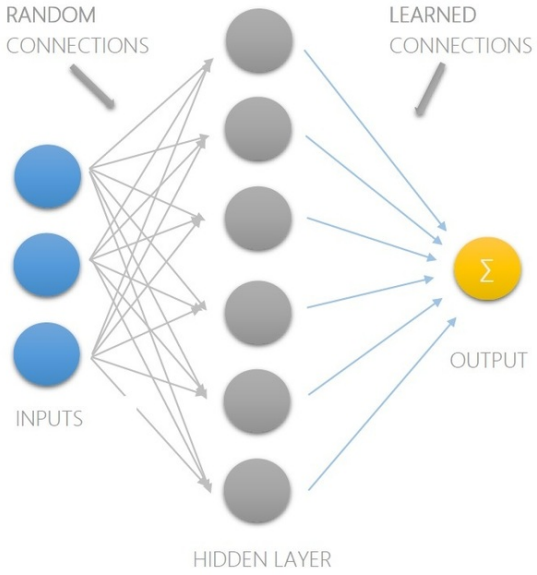


Figure 7.1: Extreme Learning Machines

Extreme Learning Machine (ELM) is a simple type of feedforward neural network (see Section (4.1.2)). It has only one hidden layer and no recurrent part. Similarly to ESN, its task performance greatly depends on random initialisation of the input weight matrix $\mathbf{W}_{in}$. Without additional tuning strategies, in some cases random generations can lead to saturations of reservoir states $\mathbf{X}(t)$, where most of the entries are equal to -1 or 1, or early convergence where $\mathbf{X}(t) = \mathbf{X}(t+1)$ when t is small.

Batch Intrinsic Plasticity (BIP) (see Neumann et al. [2011]) was introduced to adapt the activation functions so that the desired distribution for $\mathbf{X}(t)$ could be realised.

Suppose ELM has the following update equations:

$$\mathbf{X}_i = f(a_i \mathbf{W}_{in}^i \mathbf{U} + b_i) \tag{7.1.1}$$

$$\hat{\mathbf{Y}} = \mathbf{W}_{out} \mathbf{X} \tag{7.1.2}$$

where $\mathbf{U} = (U(1), U(2), ..., U(nbDataPoints)) \in \mathbb{R}^{N_u \times nbDataPoints}$ is input vector, $\mathbf{W}_{in}^i$ is the ith row of input weight matrix $\mathbf{W}_{in} \in \mathbb{R}^{N_x \times N_u}$, and $\mathbf{a} \in \mathbb{R}^{1 \times N_x}$ and $\mathbf{b} \in \mathbb{R}^{1 \times N_x}$ are scalar vectors to be determined by BIP.

Consider the activation function $f = tanh$, then $X = [-1, 1]$ is an important range where $Y = tanh(X)$ is virtually linear. Outside this range, especially when $|X| > 2$, Y will be very close to -1 or 1, which can cause saturations of $\mathbf{X}(t)$ if $\mathbf{X}_i(t) = f(\mathbf{W}_{in}^i \mathbf{U}(t))$. Fortunately, with scalar vectors $\mathbf{a}$ and $\mathbf{b}$, this situation may be mitigated.

Suppose $f = tanh$, $\mathbf{S}_0 \in \mathbb{R}^{N_x \times nbDataPoints} = \mathbf{W}_{in}\mathbf{U}$ and $\mathbf{X} \in \mathbb{R}^{1 \times N_x}$ has N_x positions for elements. Then, for the ith position, the BIP will do the following:

1. Uniform samples are drawn from $[-1, 1]$ and sorted in ascending order, the result is collected by target vector $\mathbf{T} \in \mathbb{R}^{1 \times nbDataPoints}$.

2. The ith row of $\mathbf{S}_0$ is sorted in ascending order and collected by vector $\mathbf{S}$.

Electronic copy available at: https://ssrn.com/abstract=3307566

3. Take **S** as input and **T** as output, implement a linear regression to get coefficients a_i and b_i.

Finally, iterations of i from 1 to N_x will give vectors **a** and **b**. The corresponding pseudo code can be found in the Algorithm (14) below.

Algorithm 14 BIP Training Algorithm

Require: : Import input vectors $\mathbf{u} = (u(1), u(2), ..., u(nbDataPoints))$
1: $\mathbf{S}_0 = \mathbf{W}_{in}\mathbf{U}$
2: **for** $i = 1$ to N_x **do**
3: $\mathbf{S} \leftarrow$ sorted ith row of $\mathbf{S}_0$
4: Construct concatenation $\phi = [\mathbf{S}; (1, 1, ..., 1)] \in \mathbb{R}^{2\times nbDataPoints}$
5: Draw targets $\mathbf{T} = (T(1), T(2), ..., T(nbDataPoints))$ from desired distribution
6: $\mathbf{T} \leftarrow$ sorted $\mathbf{T}$
7: $v_i = [a_i, b_i] = \mathbf{T} * pinv(\phi)$
8: **end for**
9: **Return v**

Some comments are given here:

- For f=sigmoid function, the range for uniform sampling could extend to $[-2, 2]$ due to the shape of sigmoid function.
- The input and output vectors are sorted before linear regression due to the fact that the function $\mathbf{Y} = a_i\mathbf{X} + b_i$ is set to be monotonic increasing.
- Apart from uniform distributions, the above procedure also applies to other bounded distributions. For unbounded distributions such as Gaussian, things need to be changed a little bit. Drawing normally distributed samples **T** and applying $tanh$ on it clearly no longer works since the samples are not bounded between $[-1, 1]$. At this stage, one possibility to get round would be that the normal distribution is applied on $f(\mathbf{T})$, that is, the reservoir state **X**, instead of **T**. Hence, when **X** is set to be normally distributed, f^{-1} needs to be computed. The result works as new output, say, $\mathbf{T}_2$, and linear regression can then be constructed between **S** and $\mathbf{T}_2$.

Now, this algorithm could also be used on ESN as well. Consider standard ESN with leak rate $\alpha = 1$, and the following update equation:

$$\mathbf{X}(t) = f(\mathbf{W}_{in}[1; \mathbf{U}(t)] + \mathbf{W}\mathbf{X}(t-1)) \tag{7.1.3}$$

This time, suppose design matrix $\mathbf{X} = (X(1), X(2), ..., X(nbDataPoints)) \in \mathbb{R}^{N_x \times nbDataPoints}$, there are three ways of setting $\mathbf{s}_0$:

- $\mathbf{S}_0 = f^{-1}(\mathbf{X})$
- $\mathbf{S}_0 = \mathbf{X}$
- $\mathbf{S}_0 = \mathbf{W}_{in}\mathbf{U}$

Experimentally these three settings can all improve ESN. In particular, the third one will be illustrated in detail since it generally performs better than the first two settings. This is partly because the last setting has smallest root mean square error in the desired target fitting.

Electronic copy available at: https://ssrn.com/abstract=3307566

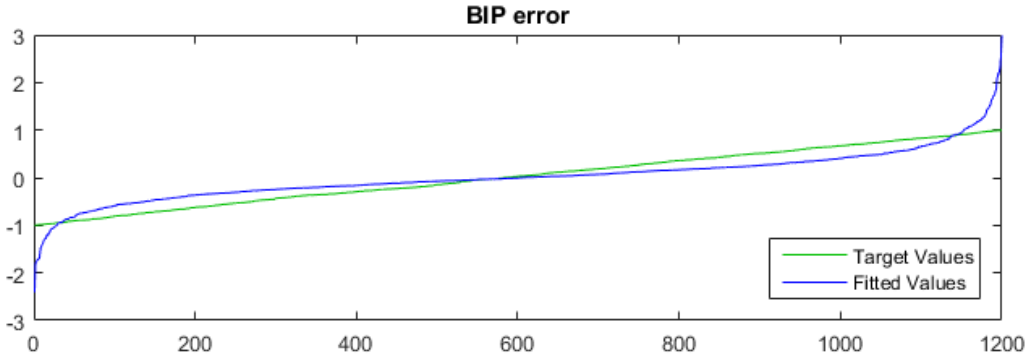


Figure 7.2: desired target fitting

Now, suppose $\mathbf{S}_0 = \mathbf{W}_{in}\mathbf{U}$, other settings kept the same as ELM, the ESN reservoir state has the following update equation:

$$\mathbf{X}(t) = f(\mathbf{a}. * \mathbf{W}_{in}\mathbf{U}(t) + \mathbf{b} + \mathbf{W}\mathbf{X}(t-1))$$

Note $\mathbf{a} \in \mathbb{R}^{N_x \times 1}$ and $\mathbf{W}_{in}\mathbf{U}(t) \in \mathbb{R}^{N_x \times 1}$, so $.*$ here means element wise multiplication. For example,

$$\begin{pmatrix} a \\ b \end{pmatrix} . * \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} ac \\ bd \end{pmatrix}$$

7.1.2 Some results

Empirically, unlike the case with original ESN, normalisation barely changes the task performance of BIP-attached ESN, while desired target range matters more. Hence, experiments are done on the stock MMM using different target ranges:

	f=tanh	f=sigmoid
Uniform Range $[-2, 2]$	$90, 0.00247, 0.00148$	$170, 0.00198, 0.00130$
Uniform Range $[-1, 1]$	$120, 0.00234, 0.00147$	$150, 0.00199, 0.00109$
Uniform Range $[-0.5, 0.5]$	$130, 0.00213, 0.00145$	$440, 0.00206, 0.00122$
Uniform Range $[0, 1]$	$110, 0.00207, 0.00134$	$450, 0.00206, 0.00125$

Similarly to the previous experiment on ESN, the triplet in the table still means N_x, training error and test error respectively. Seen from the table,

- Sigmoid in general surpasses tanh on task performance.
- For f=sigmoid, that the range $[-1, 1]$ works better than $[-2, 2]$ may be due to specific setting of $\mathbf{S}_0 = \mathbf{W}_{in}\mathbf{U}$: Since $\mathbf{X}(t) = f(\mathbf{a}. * \mathbf{W}_{in}\mathbf{U}(t) + \mathbf{b} + \mathbf{W}\mathbf{X}(t-1)) = \mathbf{a}. * \mathbf{s}_0 + \mathbf{b} + \mathbf{W}\mathbf{X}(t-1))$, $\mathbf{a}. * \mathbf{S}_0 + \mathbf{b}$ only partly contributes to calculation of $\mathbf{X}(t)$, unlike in case of ELM where $\mathbf{X}(t) = f(\mathbf{a}. * \mathbf{W}_{in}\mathbf{U}(t) + \mathbf{b})$ and $\mathbf{a}. * \mathbf{S}_0 + \mathbf{b}$ fully determines $\mathbf{X}(t)$. Moreover, $X = [-1, 1]$ is a range where $f(X)$ is steepest, hence elements of $\mathbf{X}(t)$ will be more spread out there.
- The best test error 0.00109 comes from sigmoid with uniform range $[-1, 1]$, it reduces 69% of the test error made by linear regression and 20% of the smallest test error made by original ESN. The error comparisons are displayed below:

Electronic copy available at: https://ssrn.com/abstract=3307566

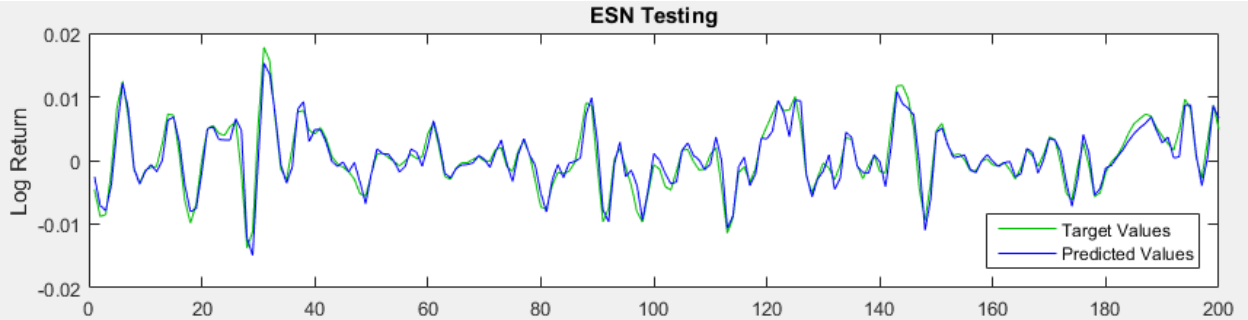


Figure 7.3: ESN on denoised stock returns test set

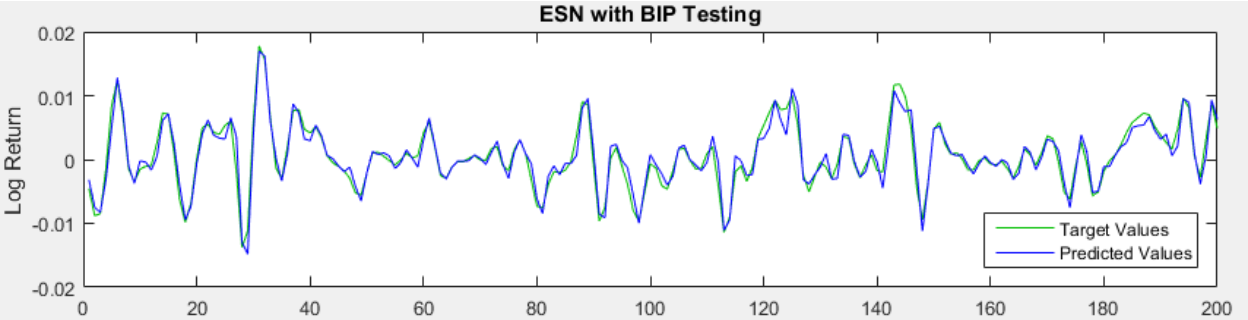


Figure 7.4: ESN with BIP on denoised stock returns test set

Also, general performance of BIP-attached ESN over 100 stocks is assessed using the following common parameter setting:

Leak Rate α	1	Spectral Radius	0.65
Reservoir size N_x	150	Activation function	sigmoid
Reservoir Size $\mathbf{W}_{in}$	$[-2, 2]$	Reservoir Weight Matrix $\mathbf{W}_{res}$	$[-0.5, 0.5]$
Reservoir Density	0.5	Regularisation term β	10^{-2}

Again, the desired target range used is $[-1, 1]$, which gives the following error table:

Errors over stocks	Average Training error(RMS)	Average Test error(RMS)
Linear Regression	0.00809	0.00799
ESN	0.00498	0.00515
ESN with BIP	0.00397	0.00459

Simple calculations show that BIP-attached ESN reduces on average 20% of training error and 11% of test error made by plain ESN. This observation underpins BIP as a promising tool for improving ESN.

7.2 Radial basis function

7.2.1 Description

Radial Basis Function (RBF) were first used to solve interpolation problem, that is, fitting a curve exactly through a set of points (see Powell [1987]). They were extended to perform the more general task of approximation (see Poggio

Electronic copy available at: https://ssrn.com/abstract=3307566

et al. [1990]).
RBF are certain type of functions whose value only depend on the distance from some center $\mathbf{c}$. Any function ϕ satisfying

$$\phi(\mathbf{X}, \mathbf{c}) = \phi(||\mathbf{X} - \mathbf{c}||)$$

is called a radial basis function. The norm here is usually taken as the Euclidean norm. Poggio et al. [1990] showed that RBF could be derived from the classical regularisation problem where some unknown function $\mathbf{Y} = f(\mathbf{X})$ can be approximated from the dataset $\{(\mathbf{X}_t, \mathbf{Y}_t)\}$ and some smoothness constraints. In terms of regression analysis (see Section (8.3.1)) the function f is the conditional expectation of $\mathbf{Y}_t$ given $\mathbf{X}_t$. That is,

$$\mathbf{Y}_t = f(\mathbf{X}_t) + \epsilon_t \ , \ E[\epsilon_t | \mathbf{X}_t] = 0$$

The regularisation problem is equivalent to the minimisation of the objective function

$$L(\hat{f}) \equiv \sum_{t=1}^{T} \left(\|\hat{\mathbf{Y}}_t - \hat{f}(\mathbf{X}_t)\|^2 + \lambda \|P\hat{f}(\mathbf{X}_t)\|^2 \right)$$

where P is a differential operator. The second term is a penalty function decreasing the smoothness of $\hat{f}$ and λ controls the tradeoff between smoothness and fit. In its most general form, the solution to the objective function is given by

$$f(\mathbf{X}) = \sum_{i=1}^{k} \alpha_i \phi(||\mathbf{X} - \mathbf{c}_i||) + p(\mathbf{X})$$

where α_i are scalar coefficients, $p(\cdot)$ is a polynomial and k is much less than the number of observations in the sample.
Girosi et al, [1990] showed that RBFs could approximate arbitrarily well any continuous function on a compact domain. As an important branch of neural networks, RBF networks have numerous applications, including system identification and non-linear chaotic time series forecasting.

It has rather distinct properties from ESN (e.g. only require small reservoir size). Micchelli [1986] showed that even though there is a large class of possible basis functions ϕ, the most common choices are the Gaussian function $e^{-\frac{x}{\beta^2}}$ and the multi-quadrics $\sqrt{x + \beta^2}$. In this section, we adopt the Gaussian kernel:

$$\mathbf{X}_i = \Phi(||\mathbf{U} - \mathbf{c}_i||) = \exp\left(\frac{-||\mathbf{U} - \mathbf{c}_i||^2}{(2\beta_i^2)}\right) \tag{7.2.4}$$

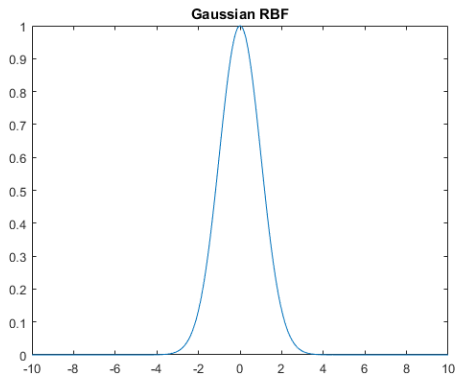


Figure 7.5: Gaussian kernel

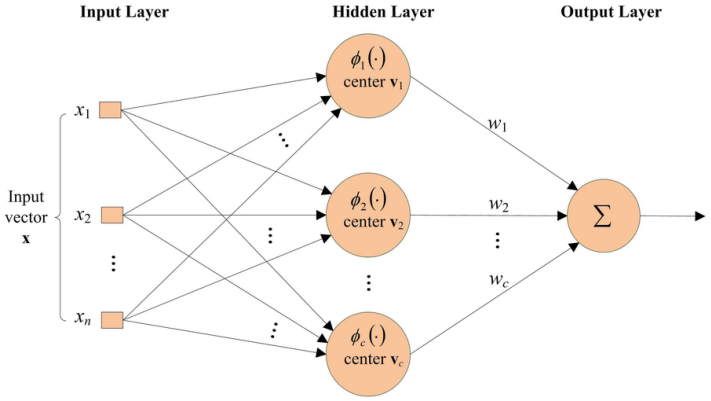


Figure 7.6: RBF

Electronic copy available at: https://ssrn.com/abstract=3307566

Essentially, each time a new input $\mathbf{U}$ is imported, it is transformed into the variable $\mathbf{X}$ using radial basis function in Equation (7.2.4). Then, the multiplication of $\mathbf{X}$ with output weight matrix $\mathbf{W}_{out}$ gives the output. This time, the output weight matrix $\mathbf{W}_{out}$ is no longer trained using a supervised ridge regression. Instead, it is tuned along with other parameters. Generally, with $\mathbf{w}$ standing for $\mathbf{W}_{out}$, the triplet $\{\mathbf{c}, \beta, \mathbf{w}\}$ needs to be tuned together using a learning algorithm called *gradient descent* (see Section (10.2.1.1)), which iteratively takes steps proportional to the negative of the current gradient of the function (see Niu et al. [2014]). The update equations are as follows:

$$w_i(n+1) = w_i(n) - \eta_1 \epsilon(n) C(n) \mathbf{x}_i(n) \tag{7.2.5}$$

$$\mathbf{c}_i(n+1) = \mathbf{c}_i(n) - \eta_2 \epsilon(n) w_i(n) C(n) \frac{\mathbf{x}_i(n)}{\beta_i(n)^2} (\mathbf{u}(n) - \mathbf{c}_i(n)) \tag{7.2.6}$$

$$\beta_i(n+1) = \beta_i(n) - \eta_3 \epsilon(n) w_i(n) C(n) \frac{\mathbf{x}_i(n)}{\beta_i(n)^3} ||\mathbf{u}(n) - \mathbf{c}_i(n)|| \tag{7.2.7}$$

Where η_i, $i = 1, 2, 3$, are learning rates which control the speed of the process. The error term $\epsilon(n)$ is the difference between target value and predicted value at time n, that is,

$$\epsilon(n) = \hat{y}(n) - y(n) = \mathbf{w}(n)\mathbf{x}(n) - y(n)$$

$C(n)$ is an additional feature called stochastic data-time effective function:

$$C(n) = \frac{1}{\tau} exp(\int_{t_0}^{n} \mu(t)dt + \int_{t_0}^{n} \sigma(t)dW_t)$$

The focus is on gradient descent algorithm itself, so for simplicity $\mu(t)$ and $\sigma(t)$ are both set to be zero. Below is the pseudo code for training RBF using gradient descent:

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 15 RBF Training Algorithm

Function$modelOutput = rbfTrain\mathbf{U}, \mathbf{y}, modelInputs$ $\{modelInputs \leftarrow N_x, \eta, \tau ...\}$
$\mathbf{c} \in \mathbb{R}^{N_u \times N_x}, \boldsymbol{\beta} \in \mathbb{R}^{N_x \times 1}, \mathbf{W}_{out} \in \mathbb{R}^{1 \times N_x}, \mathbf{yHat} \in \mathbb{R}^{1 \times nbDataPoints}$
$k = 1, error = 10^5, maxIteration = 500, tolerance = 0.008$ ← *Initializations*
while ($error >$ tolerance **and** $k < maxIteration$) **do**
  for $j = 1$ to $nbDataPoints$ **do**
    for $i = 1$ to N_x **do**
      $\boldsymbol{\phi}(i) = \exp\left(-\frac{||\mathbf{U}(:,j)-\mathbf{c}(:,i)||^2}{2\boldsymbol{\beta}(i)^2}\right)$
    end for
    $\epsilon = \mathbf{W}_{out}\boldsymbol{\phi} - \mathbf{y}(j)$
    $C = 1/\tau$
    for $i = 1$ to N_x **do**
      $w_i = \mathbf{W}_{out}(i)$
      $\mathbf{c}_i = \mathbf{c}(:,i)$
      $b_i = \boldsymbol{\beta}(i)$
      $\mathbf{W}_{out}(i) = \mathbf{W}_{out}(i) - \eta\epsilon C\boldsymbol{\phi}(i)$
      $\mathbf{c}(:,i) = \mathbf{c}(:,i) - \eta\epsilon C w_i\boldsymbol{\phi}(i)(\mathbf{u}(:,j) - \mathbf{c}_i)/b_i^2$ {$\mathbf{c}(i)$ *is a vector*}
      $\boldsymbol{\beta}(i) = \boldsymbol{\beta}(i) - \eta\epsilon C w_i\boldsymbol{\phi}(i)||\mathbf{u}(:,j) - \mathbf{c}_i||/b_i^3$
    end for
  end for
  for $j = 1$ to $nbDataPoints$ **do**
    for $i = 1$ to N_x **do**
      $\boldsymbol{\phi}(i) = \exp\left(-\frac{||\mathbf{U}(:,j)-\mathbf{c}(:,i)||^2}{2\boldsymbol{\beta}(i)^2}\right)$
    end for
    yHat$(j) = \mathbf{W}_{out}\boldsymbol{\phi}$
  end for
  $m \leftarrow nbPointsToIgnore + 1;$
  $\mathbf{y}_1 =$ (**yHat(:,m:end)** $- b)/a$ {*the first m points are discarded*}
  $\mathbf{y}_2 =$ (**y(:,m:end)** $- b)/a$ {*the y values are mapped back before calculating errors*}
  $error = \sqrt{\frac{1}{nbDataPoints}\sum_{n=1}^{nbDataPoints}(\mathbf{y}_1(n) - \mathbf{y}_2(n))^2}$
  $k = k + 1$
end while
Store updated **c**, $\boldsymbol{\beta}$ and $\mathbf{W}_{out}$
EndFunction

Note in the above algorithm:

- Suppose $N_y = 1$, then **c** is a matrix of size $N_u \times N_x$, whenever a new input $\mathbf{U}(n)$ is imported, every column of **c** is updated using $\mathbf{c}(:,i) = \mathbf{c}(:,i) - k(\mathbf{u}(:,j) - \mathbf{c}_i)$ where k is some scalar. Similarly, $\boldsymbol{\beta}$ is a vector of size $N_x \times 1$ and **w** is a vector of size $1 \times N_x$. They will be updated accordingly.
- The algorithm will terminate if either the training error is less than the predefined tolerance, or the maximum number of iterations are hit.

7.2.2 Some results

With the following parameter settings:

Electronic copy available at: https://ssrn.com/abstract=3307566

Reservoir Size N_x	15	Input size N_u	1
Normalisation Range	$[0, 1]$	Learning Rate η	0.001
Data-time Effective Function C	10	Maximum Iterations	500

and these initialisations using uniform distributions on certain ranges:

c: $[0, 1]$	β: $[0.1, 0.3]$	**w**: $[-0.1, 0.1]$

Setting the tolerance to 0.008 would give the following error table and the corresponding prediction performance:

	Training error(RMS)	Testing error(RMS)
Linear Regression	0.0315	0.032
RBF	0.008	0.00771

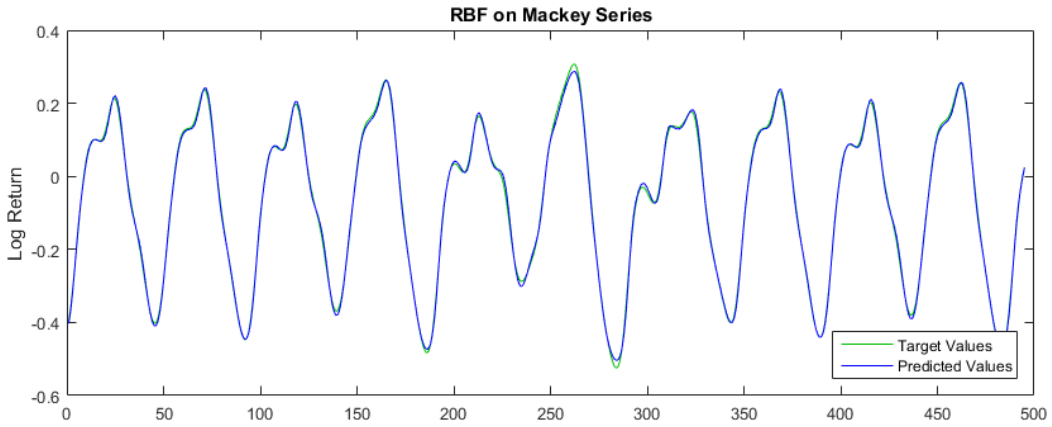


Figure 7.7: RBF on Mackey Series test set

This first experiment costs 33 seconds, and the evolution of training errors are plotted:

Electronic copy available at: https://ssrn.com/abstract=3307566

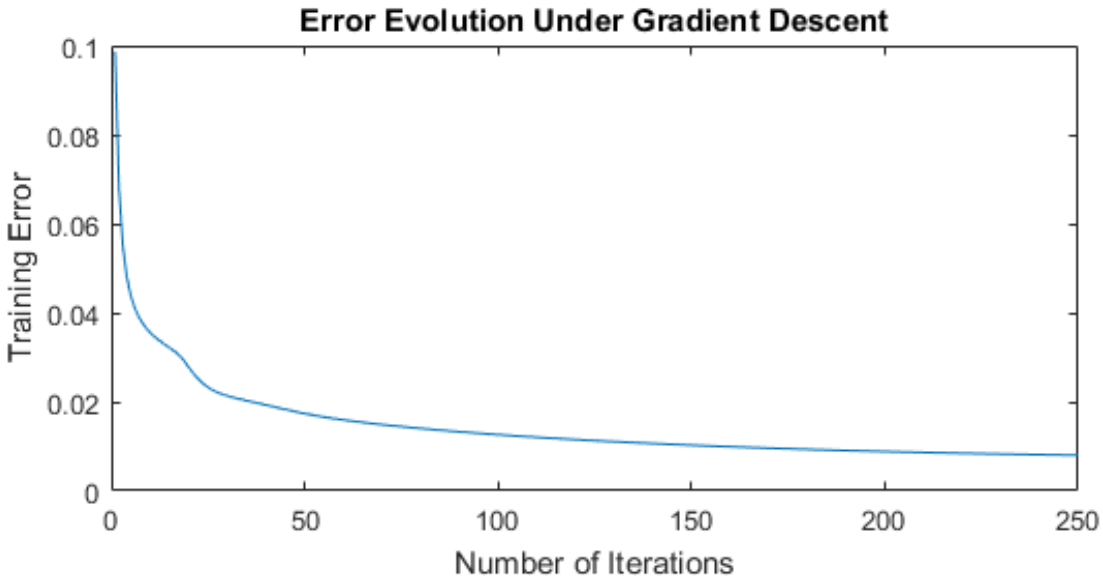


Figure 7.8: Error Evolutions

Now if the tolerance is set to be 0.007, i.e. require the training error to be no more than 0.007, then it takes 53 seconds to reach this error target:

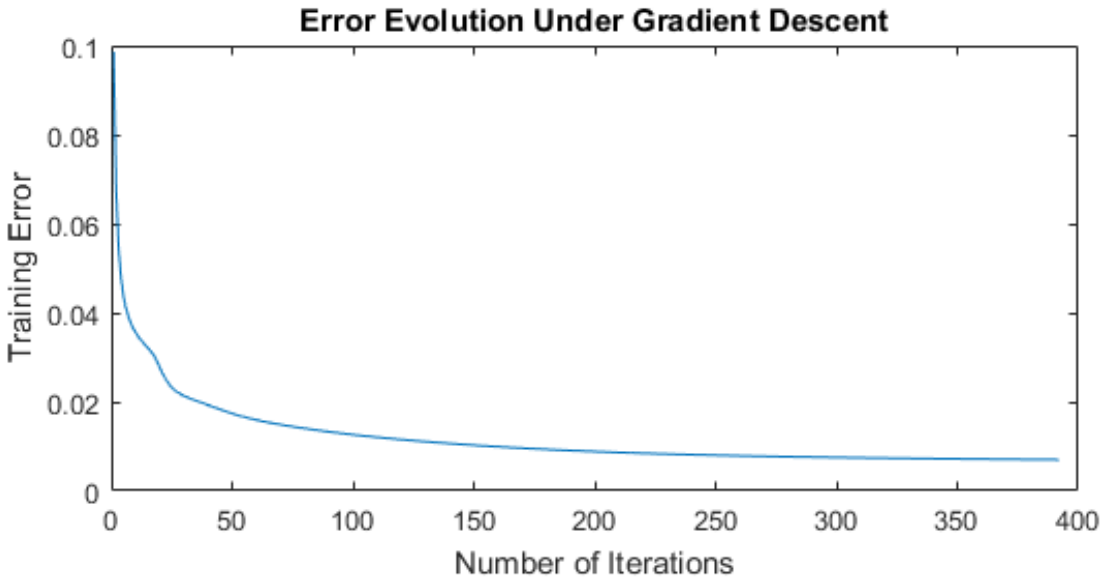


Figure 7.9: Error Evolutions

Therefore, technically it takes 20 seconds for the algorithm to get rid of the last 0.001 training error. It means that it consumes an additional 60% of the previous running time to reduce 12% of the error. Seen from the Table, the gradient descent has fast convergence rate at the beginning. However, after 200 iterations it struggles to improve just a little bit. Now, keeping the other parameters fixed (the same random initialisations are reproducible), more experiments are

Electronic copy available at: https://ssrn.com/abstract=3307566

made by varying the learning rates:

Learning Rate η	Running Time (seconds)
0.001	53
0.002	43
0.005	63
0.0005	66

In the Table (7.2.2), only a learning rate of 0.002 speeds up the process by 10 seconds. However, further increasing the learning rate ($\eta = 0.005$) no longer helps.
In addition to learning rate, the evolvements of $\mathbf{w}$, $\boldsymbol{\beta}$ and $\mathbf{c}$ are also of interest. In the original case where tolerance=0.008 and learning rate=0.001, the final form of $\boldsymbol{\beta}$ is displayed in Figure (7.10).

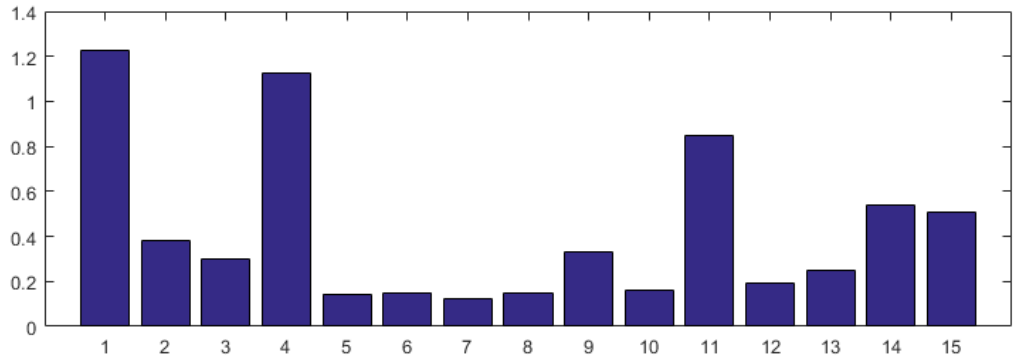


Figure 7.10: bar plot of $\boldsymbol{\beta}$

Since $N_x = 15$, each bar represents each element of $\boldsymbol{\beta}$. Note that $\boldsymbol{\beta}$ is initialised to be within range $[0.1, 0.3]$. But as seen from the plot, about $\frac{1}{3}$ of the elements go far beyond its upper bound 0.3, in particular, the highest value is 1.23, which is about 4 times 0.3, and the lowest value is 0.12. Hence, if the initialisation range for $\boldsymbol{\beta}$ is expanded to $[0, 1]$ or $[0.12, 1.23]$, it is possible that the running time will be further reduced. Empirically however, neither case manages to decrease the original running time. Nevertheless, if the learning rate is adjusted to 0.002, all three cases achieve a reduction of 10 seconds on average. On the other hand, similar experimental results were obtained to the case of $\mathbf{w}$ and $\mathbf{c}$.

In general, the learning rate is a delicate parameter in machine learning. If it is set too small, it will take too much time to converge. On the other hand, if it is greater than needed, it may never converge and even diverge. See the Figure (7.11) for intuition.

Electronic copy available at: https://ssrn.com/abstract=3307566

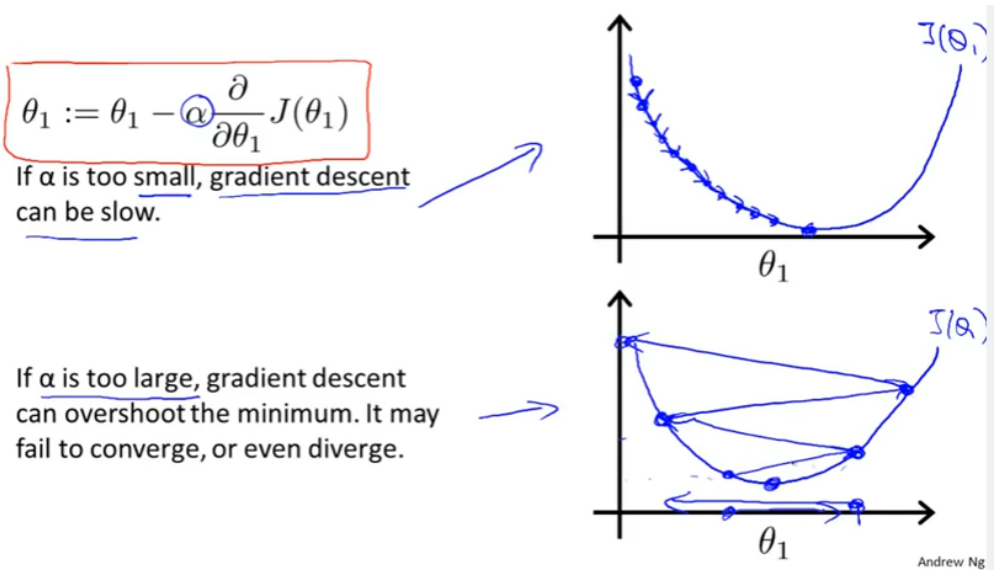

$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$
If α is too small, gradient descent can be slow.
If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.
$J(\theta_1)$
θ_1
Andrew Ng

Figure 7.11: Gradient Descent-Andrew Ng

At the same time, although for gradient descent, initialisations are not required to be close-to-optimal, it has to be within a certain range for the algorithm to reach the target error within reasonable time. Ideally, the gradient descent works best on convex functions. See the Figure (7.12) for intuition. If the function has multiple peaks or several local minima, given different initialisations, gradient descent may end up finding the global minimum or being trapped in a local minimum.

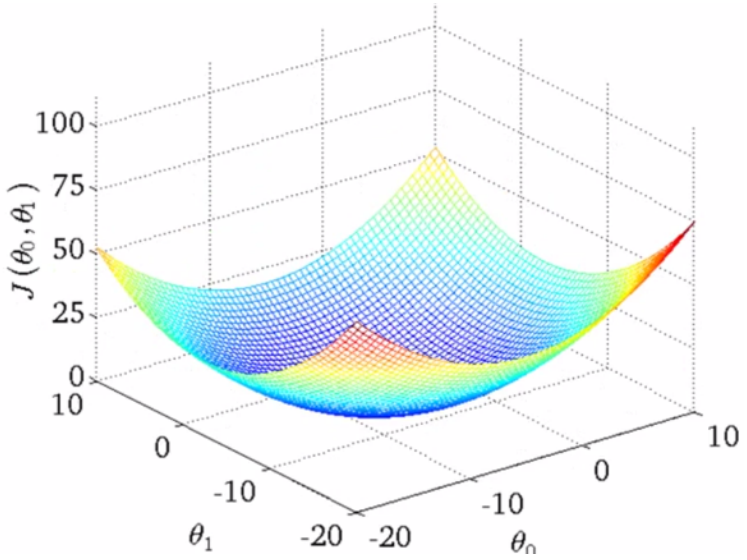


Figure 7.12: Convex Functions

In summary, this section introduced a new parameter optimising method. The main advantage of gradient descent is that one can specify the desired error level beforehand so that task running time could be allocated more efficiently according to different accuracy requirements. At the same time, however, it has several limitations:

Electronic copy available at: https://ssrn.com/abstract=3307566

- The setting of learning rate largely depends on experience. Inappropriate setting can make it impossible to reach the target error level within reasonable time.
- Gradient descent is relatively slow close to the minimum: technically, its asymptotic rate of convergence is inferior to many other methods.
- Gradient descent fails confronting non-differentiable functions. Thus these functions may need to be bounded by a smooth function for gradient descent to work.

7.3 Recurrent RBF network

7.3.1 Standard RRBFN

Recall from Section (7.2), the radial basis function (RBF) model has the form given in Equation (7.2.4). This model is now featured with recurrent structure so that it turns into an advanced structure called Recurrent Radial Basis Function Network(RRBFN) which has the following form:

$$\begin{aligned}\mathbf{x}_i(n) &= \Phi\big(||\mathbf{U}(n) - \mathbf{c}_1^i||, ||\mathbf{X}^i(n-1) - \mathbf{c}_2^i||\big) \\ &= \exp\big(-(||\mathbf{u}(n) - \mathbf{c}_1^i||^2 + ||\mathbf{X}_i(n-1) - \mathbf{c}_2^i||^2)/(2\beta_i^2)\big)\end{aligned} \tag{7.3.8}$$

This time, the function values depend on two inputs instead of one. Two 'distances' from the two center vectors determine the output value. More specifically, its update equation is:

$$\tilde{\mathbf{X}}_i(n) = \exp\big(-\alpha||\mathbf{W}_{in}^i - \mathbf{U}(n)||^2 - \beta||\mathbf{W}^i - \mathbf{X}(n-1)||^2\big), \quad i = 1, ..., N_x$$

where $\tilde{\mathbf{x}}_i(n)$ is the ith element of vector $\tilde{\mathbf{X}}(n) \in \mathbb{R}^{N_x}$; $\mathbf{W}_{in}^i$ and $\mathbf{W}^i$ are ith row of $\mathbf{W}_{in}$ and $\mathbf{W}$ respectively, α and β are scalar parameter which absorb the information of β_i in Equation (7.2.4).

7.3.1.1 Pseudo code

The following is the pseudo code for training RRBFN:

Algorithm 16 RRBFN Training Algorithm

Function$rrbfTrain\mathbf{u}, \mathbf{y}, modelInputs$ {$modelInputs \leftarrow N_x, \mathbf{W}_{in}, \mathbf{W}, \mathbf{X}_0$ *etc.*}
$\mathbf{X} \in \mathbb{R}^{N_x \times nbDataPoints} \leftarrow$ *Design Matrix*
for $i = 1$ to $nbDataPoints$ **do**
  for $j = 1$ to N_x **do**
    $\mathbf{X}(j,i) = \exp\big(-\alpha||[\mathbf{W}_{in}(j,:)]^T - \mathbf{U}(:,i)||^2 - \beta||[\mathbf{W}(j,:)]^\top - \mathbf{X}(:,i-1)||^2\big)$;
  end for
  $\mathbf{z}(:,i) = [1; \mathbf{u}(:,i); \mathbf{X}(:,i)]$; {*Vertical Concatenation*}
end for
$m \leftarrow nbPointsToIgnore + 1$;
$S = \mathbf{z}(:, m : end)$;
$D = \mathbf{y}(:, m : end)$;
$\mathbf{W}_{out} = DS^T pinv(SS^\top + \lambda I)$; {*Ridge Regression, 'pinv' is pseudo inverse*}
EndFunction

Electronic copy available at: https://ssrn.com/abstract=3307566

7.3.1.2 Parameter tuning

Again, the Mackey series is used as a benchmark chaotic time series to tune the parameters of RRBFN. Compared to other chaotic time series, even noisy, it is relatively smooth and tractable. Therefore, if a machine learning model cannot succeed on training the Mackey series, then it is unlikely to have strong potential in predicting chaotic time series. Most importantly, the Mackey Series could be used to find priority of parameters. In general,

- An increase in N_u will monotonically decrease both training error and test error to a certain level. However it is not recommended to set a high value for N_u since it increases the chance of overfitting.
- W barely changes results.
- Other parameters give no obvious trends and thus need to be tuned one at a time.

It is worth mentioning that α and β have similar positions in the model, they both are scalar factors of a certain 'distance'. That is,

- α is the distance between ith row of $\mathbf{W}_{in}$ and input vector $\mathbf{u}(n)$.
- β is the distance between ith row of $\mathbf{W}$ and reservoir state $\mathbf{x}(n-1)$.

Therefore, these parameters should be treated as a pair.
First of all, assume α and β have equal importance. Start from $\alpha = \beta = 0.1$, a monotonic increase in both values until $\alpha = \beta = 0.9$ gives a general decrease in errors. In particular, $\alpha = \beta = 0.6$ is chosen to be a robust set since there is a huge drop in errors when $\alpha = \beta = 0.5$ is changed to $\alpha = \beta = 0.6$, and also this is the first time when test error is less than 10^{-3}. Nevertheless, $\alpha = \beta = 0.9$ is robust too, for it produces smallest test error when α and β are set equal.

Secondly, it remains to be seen which one has more impact on final result. Fixing other parameters, the following statistics are obtained:

α	0.6	0.6	1
β	0.6	1	0.6
Test error(RMSE)	0.000737	0.000684	0.000852

From this table, it can be observed that a 66% change in β leads to a 7% decrease in the test error, while 66% change in α results in 15.6% increase in the test error, which indicates that generally α is more influential on the error term. Then fixing $\alpha = 0.6$, careful tuning on β gives the final pair:

α	0.6
β	0.9
Test error(RMSE)	0.000669

Similarly, it is possible that some other parameters are also strongly inter-connected and should be tuned together. However, identification of this requires additional expertise and deep analysis on the behaviour of parameters.

Finally, with cautiously tuned parameters below,

$[\alpha,\beta]$	$[0.6,0.9]$	Spectral Radius	1.25
Reservoir size N_x	10	Input Size N_u	1
Reservoir Size $\mathbf{W}_{in}$	$[-1,1]$	Regularization Term	10^{-8}

Without using data normalisation, the errors are as displayed:

Electronic copy available at: https://ssrn.com/abstract=3307566

	Training error(RMS)	Test error(RMS)
Linear Regression	0.0315	0.032
RRBF	0.000705	0.000648

From now on it is assumed that 0.000648 is the smallest test error that can be achieved by standard RRBFN with $N_u = 1$. This important result will be cited later for further comparison and evaluation with new methods.

7.3.2 Randomly parametrised RRBFN

Based on the conjecture that each entry of input vector $\mathbf{U}(n)$ or reservoir states $\mathbf{X}(n)$ can contribute differently to the new reservoir state $\mathbf{X}(n+1)$, modifications on distance computation are made hoping to take advantage of this hypothesis. For example, if $\mathbf{U}(n) = (U_1, U_2, U_3, U_4, U_5)^\top$ is a vector representing the log returns of the past 5 days, then U_1 could have more impact on $\mathbf{U}(n+1)$ than U_5 does. Therefore, when computing $\mathbf{W}_{in}^i - \mathbf{U}(n)$, the resultant vector $[0.1, 0, 0, 0, 0]^\top$ should be distinguished from $[0, 0, 0, 0, 0.1]$. If one favours small values of the norm $||\mathbf{W}_{in}^i - \mathbf{U}(n)||$, then the latter vector is more desirable than the first one since U_1 matters more than U_5. However, in the sense of previous Euclidean norm, the two vectors have exactly the same norm. Therefore, *weights* may be assigned to different positions within a vector to unearth and make use of the priority of importance.

However, usually the order of importance within $\mathbf{U}(n)$ is unknown and it may be constantly changing. If a specific order list is presumed and widely used over different time periods and stocks, generality of the model may be impaired. Hence, N_u being the length of vector $\mathbf{U}(n)$, a randomly assigned weight distribution is generated following the steps below:

- Draw N_u samples from standard uniform distribution $\mathrm{U}(0, 1)$.
- Calculate the sum of N_u samples, call it s;
- Divide each sample by s to get a probability distribution $\mathbf{p}$;
- Multiply each element of $\mathbf{p}$ by N_u to get final weight vector $\mathbf{w}_1$;

In the situation of equal weight distribution, all elements of $\mathbf{p}$ are the same and equal to $\frac{1}{N_u}$, and the weight vector $\mathbf{w}_1 = N_u * \mathbf{p}$ consists of all ones. This special case is equivalent to the previous standard RRBFN, where distances are measured in Euclidean norm.
The above justifications and calculations apply similarly to the norm $||\mathbf{W}^i - \mathbf{X}(n-1)||$, with N_u changed into N_x.

Fixing the optimal parameter settings for standard RRBF, iterations over different weight vector $\mathbf{w}_1$ show that the random parametrised RRBF is relatively unpredictable in improving standard RRBF. Sometimes, it could reduce the test error by 20%; On the other hand, with some extreme distribution of $\mathbf{w}_1$, this could be the other way round. Therefore, its performance largely depends on initialisation of $\mathbf{w}_1$ and is relatively unstable.

Below is a case where $\mathbf{w}_1$ is randomly initialised using rand('seed',9). In this case, randomly parametrised RRBF reduces 17% of the test error made by standard RRBF:

	Training error(RMS)	Test error(RMS)
Linear Regression	0.0315	0.032
Standard RRBF	0.000705	0.000648
Randomly Parametrized RRBF	0.000591	0.000539

Electronic copy available at: https://ssrn.com/abstract=3307566

7.3.3 Directional parametrised RRBFN

Due to the fact that the performance of randomly parametrised RRBFN greatly relies on random generation of the weight vector $\mathbf{w}_1$, one may want to alleviate, or get rid of, the random factor by generating $\mathbf{w}_1$ using non-random rules. This type of network is called directional parametrised RRBFN. It has been paid attention since in the previous experiment, some distributions of $\mathbf{w}_1$ do lead the parametrised RRBFN to a better performance than that of standard RRBFN.
In our trial, elements of $\mathbf{w}_1$ are set to be proportional to magnitudes of elements in vector $\mathbf{W}_{in}^i - \mathbf{u}(n)$. For instance, if $\mathbf{W}_{in}^i - \mathbf{U}(n) = (1,3)^\top$, then the corresponding $\mathbf{p}$ and $\mathbf{w}_1$ would be $(0.25, 0.75)^\top$, $(0.5, 1.5)^\top$ respectively. Note that only the magnitude matters. So if $\mathbf{W}_{in}^i - \mathbf{U}(n) = (-1,3)$, the corresponding $\mathbf{w}_1$ would still be $(0.25, 0.75)^\top$. Under this rule, the vector difference between $\mathbf{W}_{in}^i$ and $\mathbf{U}(n)$ is amplified. For example, if $\mathbf{W}_{in}^i - \mathbf{U}(n) = (1,3)^\top$ holds, then

- Euclidean norm would give $\sqrt{1^2 + 3^2} = \sqrt{10}$
- Under new rule, the norm is $\sqrt{0.5 * 1^2 + 1.5 * 3^2} = \sqrt{14}$

Empirically, with proper parameter settings:

- α=0.2, β=0.8
- Spectral Radius=0.9

Directional parametrised RRBF is able to improve the best standard RRBF by a meaningful amount. Refer to the following table for details:

	Training error(RMS)	Test error(RMS)
Linear Regression	0.0315	0.032
Standard RRBF	0.000705	0.000648
Randomly Parametrized RRBF	0.000591	0.000539
Directional Parametrized RRBF	0.000548	0.000479

From the table, it can be seen that the directional parametrised RRBF successfully reduces 26% of the test error made by standard RRBF. This observation is encouraging since it shows that directional parametrised RRBF may have potential to *steadily* improve RRBF with general stock data.

7.3.4 Empirical results

Firstly, the test is done on a specific stock. For consistency, stock 'MMM' is again used. After plugging in appropriate parameters, the errors are displayed below:

	Training error(RMS)	Test error(RMS)
Linear Regression	0.00472	0.0036
Standard RRBF	0.00290	0.00173
Directional Parametrized RRBF	0.00269	0.00153

Again, the above errors in the table are smallest errors (especially test error) attainable using corresponding methods. This time, the directional parametrised RRBF decreases 12% of the test error made by standard RRBF, which is a good start.

Now, with a common parameter setting, data are collected across 100 stocks:

Electronic copy available at: https://ssrn.com/abstract=3307566

	Average Training error(RMS)	Average Test error(RMS)
Linear Regression	0.00809	0.00799
Standard RRBF	0.00531	0.00598
Directional Parametrized RRBF	0.00519	0.00532

Several comments are given below:

- To some extent, it is proved that the directional parametrised RRBF is capable of improving standard RRBF in terms of both training and test error.
- On average, the directional parametrised RRBF decreases 11% of the test error made by standard RRBF. However, it takes around 1 minute to run the program, while standard RRBF costs 4 seconds. Nevertheless, the time is still acceptable.
- In general, an significant increase in N_x does not effectively reduce test error. This might be one of the main differences between ESN and RRBF: For RRBF, N_x is usually chosen to be less then 15, while in ESN, N_x can be in hundreds, thousands, even in ten thousands. This structural difference may be part of the reason why RRBF has average test error which is 17% higher than that of ESN.

7.3.5 Unsupervised learning for RRBFN

There are many unsupervised learning algorithms for training RBF type neural networks. One of them is called Self Organising Map (SOM) (see Lukosevicius [2010]). See Figure (7.13).

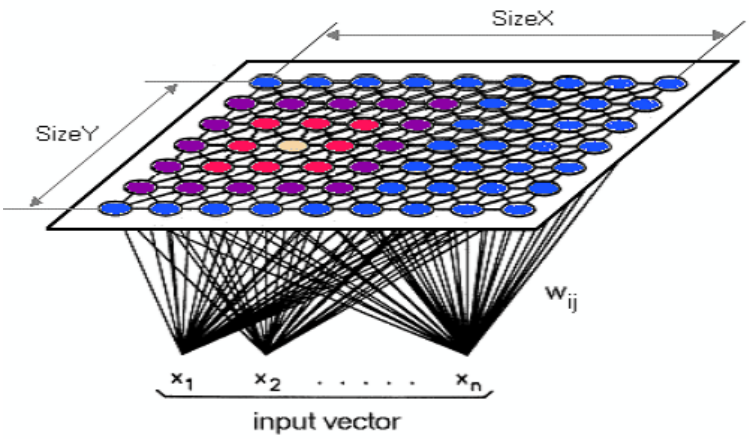


Figure 7.13: Self-Organizing Maps

Its update equations are:

$$\mathbf{W}_{in}^i(n+1) = \mathbf{W}_{in}^i(n) + \eta(n)h(i,n)(\mathbf{U}(n) - \mathbf{W}_{in}^i(n)) \tag{7.3.9}$$
$$\mathbf{W}^i(n+1) = \mathbf{W}^i(n) + \eta(n)h(i,n)(\mathbf{X}(n) - \mathbf{W}^i(n)) \tag{7.3.10}$$

where $\eta(n)$ is the learning rate and $h(i,n)$ is called learning gradient distribution function:

$$h(i,n) = exp(\frac{-d_h(i,bmu(n))^2}{b_h(n)^2}) \tag{7.3.11}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

In the above equation, bmu(n)=argmax$_i$($\mathbf{x}_i$(n)) is the index of the 'best matching unit'(BMU), $d_h(i,j)$ is the distance between units with indices i and j in the additionally defined topology for reservoir units. In our case, $d_h(i,j)$ is:

- either the Euclidean distance between $\mathbf{W}_{in}^i$and $\mathbf{W}_{in}^{bmu(n)}$.
- or the Euclidean distance between $\mathbf{W}_i$ and $\mathbf{W}_{bmu(n)}$.

Practically, every time a new input $\mathbf{U}(n)$ is imported, firstly the reservoir state $\mathbf{X}(n)$ is updated using $\mathbf{U}(n)$ and $\mathbf{X}(n-1)$:

$$\tilde{\mathbf{X}}_i(n) = exp\big(-\alpha||\mathbf{W}_{in}^i - \mathbf{U}(n)||^2 - \beta||\mathbf{W}^i - \mathbf{X}(n-1)||^2\big), \quad i = 1, ..., N_x \tag{7.3.12}$$

Then, Equations (7.3.9) and (7.3.10) are used to update rows of $\mathbf{W}_{in}$ and $\mathbf{W}$ adaptively. Meanwhile, this updating effect is controlled by a scaler called *learning rate* η. Empirical values for learning rate are $0.1, 0.01, 0.001$. In some cases, under certain criteria, the value of learning rate itself can dynamically change during the training process to better fit the dataset. But for simplicity, we use fixed value for η here.

Take Equation (7.3.9) for instance: consider input weight matrix $\mathbf{W}_{in} \in \mathbb{R}^{N_x \times N_u}$, with every row being a *center* vector to the new input $\mathbf{U}(n)$, then there are in total N_x center vectors. Hence, N_x is supposed to be the estimated number of *independent* center vectors. Ideally,

- These center vectors should be far from each other in Euclidean distance.
- Every new input $\mathbf{U}(n) \in \mathbb{R}^{N_u}$ should close to all of the center vectors, i.e. all the rows of $\mathbf{W}_{in}$.

However, due to the fact that $\mathbf{W}_{in}$ and $\mathbf{W}$ are randomly initialised, their rows may not function as suitable center vectors that perfectly match input dataset. Therefore, they need to be adaptively updated during the training process.

Whenever a new input $\mathbf{U}(n)$ is imported, its distances to every rows of $\mathbf{W}_{in}$ are computed. Among all N_x distances, the best matching unit (BMU) is the row index of $\mathbf{W}_{in}$ which is closest to $\mathbf{U}(n)$. Then, assuming $bmu = k$, the distances from the kth row to every other rows of $\mathbf{W}_{in}$ are calculated and the results are summarised in the term $h(i,n)$, where i means ith row and n indicates the time point. Specifically, it has the following expression:

$$h(i,n) = exp\big(\frac{-||\mathbf{W}_{in}^i(n) - \mathbf{W}_{in}^k(n)||^2}{(2\sigma^2)}\big)$$

Here the setting of σ varies from case to case, usually it is the volatility of the variable of interest.

Now, look back to Equation (7.3.9): if $\mathbf{U}(n)$ is closest to a center $\mathbf{W}_{in}^k(n)$, then $\mathbf{W}_{in}^k(n)$ becomes $\mathbf{W}_{in}^{bmu}(n)$. If $\mathbf{W}_{in}^{bmu}(n)$ is far from other center vectors, then the value of $h(i,n)$ should be small since the norm $||\mathbf{W}_{in}^i(n) - \mathbf{W}_{in}^{bmu}(n)||$ would be large. At the same time, if $\mathbf{U}(n)$ does not deviate too much from all the center vectors, $\mathbf{U}(n) - \mathbf{W}_{in}^i(n)$ would be relatively small for all i. Hence, the term $\eta(n)h(i,n)(\mathbf{U}(n) - \mathbf{W}_{in}^i(n))$ as a whole would be small if the learning rate $\eta(n)$ is selected and fixed. As a matter of fact, this is a desirable situation and $\mathbf{W}_{in}$ does not need to be modified much.

Moreover, it holds true that for $i = 1...N_x$,

$$||\mathbf{U}(n) - \mathbf{W}_{in}^{bmu}(n)|| <= ||\mathbf{U}(n) - \mathbf{W}_{in}^i(n)|| \tag{7.3.13}$$

$$1 = h(bmu(n), n) >= h(i,n) > 0 \tag{7.3.14}$$

However, it is not clear if $h(bmu(n),n)(\mathbf{U}(n) - \mathbf{W}_{in}^{bmu}(n)) >= h(i,n)(\mathbf{u}(n) - \mathbf{W}_{in}^i(n))$. That is, we are not aware if $\mathbf{W}_{in}^{bmu}$ moves greater distance towards $\mathbf{U}(n)$ than other rows do. Practically, it turns out that most of the time $\mathbf{W}_{in}^{bmu}$

Electronic copy available at: https://ssrn.com/abstract=3307566

tends to move faster due to the exponential decay of $h(i,n)$. Nevertheless, there exist other models such as Neural Gas (NG) (see Lukosevicius [2010]) that require $h(bmu(n),n)=0$. The above explanation similarly applies to Equation (7.3.10). Only the 'distance' involved changes.

Furthermore, due to the fact that the rows of $\mathbf{W}_{in}$ is always compared to $\mathbf{U}(n)$, in some cases the first N_x training inputs $\mathbf{U}(1)$, $\mathbf{U}(2)$,...,$\mathbf{U}(N_x)$ are used to initialise $\mathbf{W}_{in}$:

$$\mathbf{W}_{in} = \begin{bmatrix} — & U(1)^\top & — \\ — & U(2)^\top & — \\ & \vdots & \\ — & U(N_x)^\top & — \end{bmatrix} \in \mathbb{R}^{N_x \times N_u}$$

7.3.5.1 Pseudo code

The following is the pseudo code for training RRBFN with SOM:

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 17 RRBFN with SOM Training Algorithm

Function$rrbfSOMTrain\mathbf{U}, \mathbf{y}, modelInputs$ {$modelInputs \leftarrow N_x, \mathbf{W}_{in}, \mathbf{W}, \mathbf{X}_0$ *etc.*}
$\mathbf{X} \in \mathbb{R}^{N_x \times nbDataPoints} \leftarrow$ *Design Matrix*
for $i = 1$ to $nbDataPoints$ **do**
  for $j = 1$ to N_x **do**
    $\mathbf{X}(j,i) = exp(-\alpha||[\mathbf{W}_{in}(j,:)]^T - \mathbf{U}(:,i)||^2 - \beta||[\mathbf{W}(j,:)]^T - \mathbf{X}(:,i-1)||^2)$; {*Finally update* $\mathbf{x}(n)$}
  end for
  $\mathbf{z}(:,i) = [1; \mathbf{U}(:,i); \mathbf{X}(:,i)]$; {*Vertical Concatenation*}
  $BMU_1 = ||\mathbf{W}_{in}(1,:)^T - \mathbf{U}(:,i)||$;
  $Index_1 = 1$;
  for $j = 2$ to N_x **do**
    $Test = ||\mathbf{W}_{in}(j,:)^T - \mathbf{U}(:,i)||$; {*To find* BMU_1}
    if $Test < BMU_1$ **then**
      $BMU_1 = Test$;
      $Index_1 = j$;
    end if
  end for
  $BMU_2 = ||\mathbf{W}(1,:)^T - \mathbf{X}(:,i)||$;
  $Index_2 = 1$;
  for $j = 2$ to N_x **do**
    $Test = ||\mathbf{W}(j,:)^T - \mathbf{X}(:,i)||$; {*To find* BMU_2}
    if $Test < BMU_2$ **then**
      $BMU_2 = Test$;
      $Index_2 = j$;
    end if
  end for
  for $k = 1$ to N_x **do**
    $\mathbf{W}_{in}(k,:) = \mathbf{W}_{in}(k,:) + \eta([\mathbf{U}(:,i)]^\top - \mathbf{W}_{in}(k,:))exp(-||\mathbf{W}_{in}(k,:) - \mathbf{W}_{in}(Index_1,:)||^2/(2\sigma^2))$; {*Update* $\mathbf{W}_{in}$ *and* $\mathbf{W}$}
    $\mathbf{W}(k,:) = \mathbf{W}(k,:) + \eta([\mathbf{x}(:,i)]^\top - \mathbf{W}(k,:))exp(-||\mathbf{W}(k,:) - \mathbf{W}(Index_2,:)||^2/(2\sigma^2))$;
  end for
end for
$m \leftarrow nbPointsToIgnore + 1$;
$S = \mathbf{z}(:, m:end)$;
$D = \mathbf{y}(:, m:end)$;
$\mathbf{W}_{out} = DS^T pinv(SS^T + \lambda I)$; {*Ridge Regression, pinv is pseudo inverse*}
EndFunction

7.3.5.2 Empirical results

Fix the optimal parameter setting for standard RRBF on stock 'MMM',

$[\alpha,\beta]$	$[0.6,1]$	Spectral Radius	1.25
Reservoir size N_x	12	Normalization Range	$[0,1]$
Reservoir Size $\mathbf{W}_{in}$	$[-1,1]$	Regularisation Term	10^{-8}

and only add unsupervised update equation for $\mathbf{W}_{in}$ and $\mathbf{W}$ (see Equations (7.3.9) and (7.3.10)) with learning rate $\eta = 0.01$ to the algorithm, the corresponding errors are shown below:

Electronic copy available at: https://ssrn.com/abstract=3307566

	Training error(RMS)	Test error(RMS)
Linear Regression	0.00472	0.0036
Standard RRBF	0.00290	0.00173
Standard RRBF with SOM	0.00258	0.00163

Seen from the table, with adjustment of $\mathbf{W}_{in}$ and $\mathbf{W}$, the SOM algorithm manages to reduce 6% of the test error made by standard RRBF. Please see the evolution of input weight matrix $\mathbf{W}_{in}$ below:

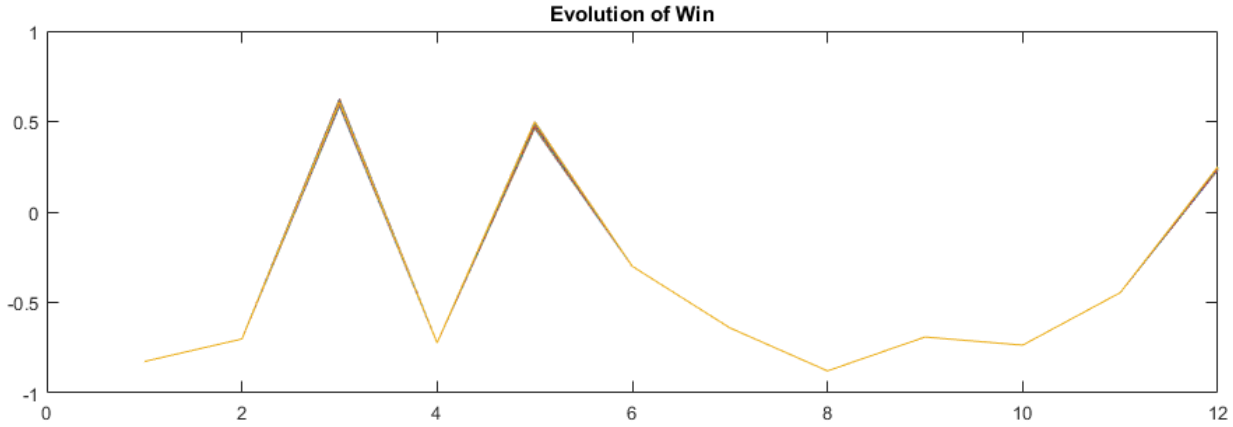


Figure 7.14: Evolution of $\mathbf{W}_{in}$ during training

Note that $\mathbf{W}_{in}$ has dimension $N_x \times N_u = 12 \times 1$. In the above figure, since $N_x = 12$, the x-axis has 12 discrete points representing 12 positions in the vector $\mathbf{W}_{in}$. For each position, the corresponding y-value is the element value on that position. In theory, there supposed to be 1200 curves, with each curve shown in unique color. However, most of the curves coincide with each other and one can hardly tell one from another. This figure indicates that $\mathbf{W}_{in}$ has not changed a lot during the process. With a closer look at the figure, the 3th, 5th and 12th element may be found to evolve a bit during the training process.
Note that it is only in the training process that SOM is applied, the test set is always left untouched. At the end of the training process, the evolved $\mathbf{W}_{in}$ and $\mathbf{W}$ are used on test set. In order to reduce the chance of overfitting, no further evolution is undertaken at test stage.

Similar to $\mathbf{W}_{in}$, the evolution of $\mathbf{W}$ is also of interest. However, since $\mathbf{W}$ has $2D$ dimension of $N_x \times N_x$, it is difficult to visualise its adaptation process. Nevertheless, the initial and final formations of the reservoir weight matrix $\mathbf{W}$ are plotted:

Electronic copy available at: https://ssrn.com/abstract=3307566

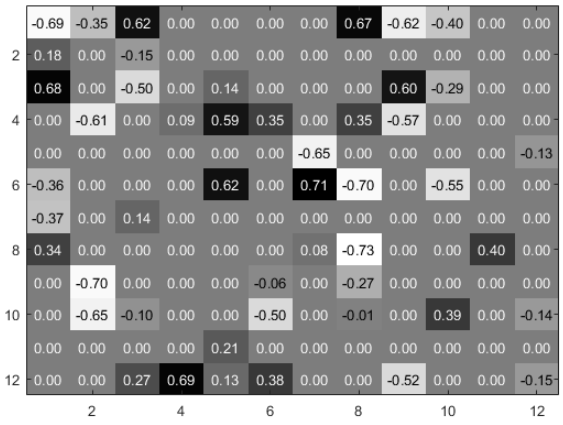


Figure 7.15: Initial W

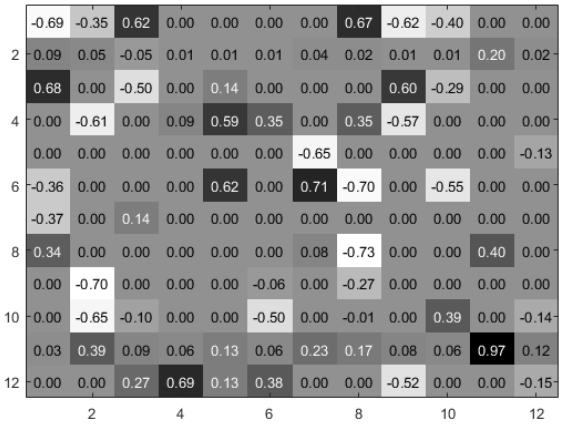


Figure 7.16: Evolved W

Note that the deeper the colour, the higher the value. In the initial $\mathbf{W}$, the matrix is supposed to be sparse so that only 40% of them are non-zero. Comparing two figures, it could be seen that only 2nd and 11th row changes by a meaningful amount. This is partly because originally $\mathbf{W}$ is already relative close to optimal setting.

Now, keeping the other parameters fixed, if the range of $\mathbf{W}_{in}$ changes to $[-0.5, 0.5]$, then the errors generated are as follows:

	Training error(RMS)	Test error(RMS)
Linear Regression	0.00472	0.0036
Standard RRBF	0.00332	0.00213
Standard RRBF with SOM	0.00319	0.00189

The corresponding evolution of $\mathbf{W}_{in}$ is also displayed:

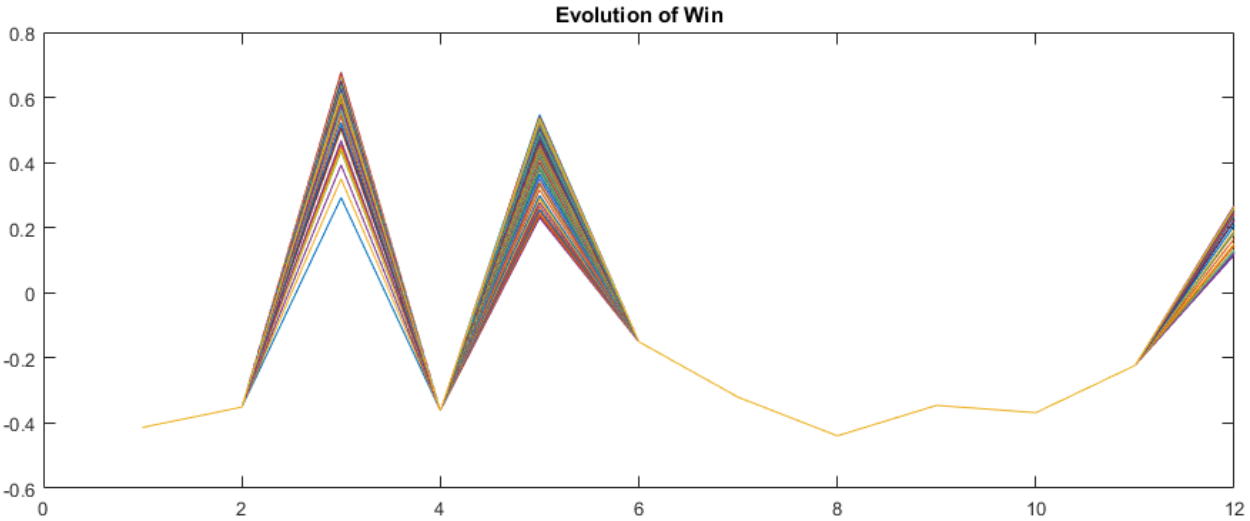


Figure 7.17: Evolution of $\mathbf{W}_{in}$

Electronic copy available at: https://ssrn.com/abstract=3307566

Several comments are now presented:

- Seen from the table, SOM successfully reduces 11% of the error made by standard RRBF. The error reduction doubles compared to the previous case.
- The figure shows a relative large evolution on the 3th, 5th and 12th rows. It looks as if SOM tries to 'rectify' these rows of W_{in} and lead them to the right track. Generally, in those much more complex tasks, where optimal parameter settings are hard to find, SOM can guide the system adaptively in the right direction.

Now, the general performance of SOM is tested on 100 stocks. The parameter settings are kept exactly the same as the previous one used for standard RRBF tests across 100 stocks. The resultant errors are as follows:

	Average Training error(RMS)	Average Test error(RMS)
Linear Regression	0.00809	0.00799
Standard RRBF	0.00531	0.00598
Standard RRBF with SOM	0.00509	0.00571

Seen from the table, on average SOM reduces test errors by 5%, which is half of the previous 11% reduction on a specific stock. Moreover, the error percentage reduction distribution across 100 stocks is also displayed:

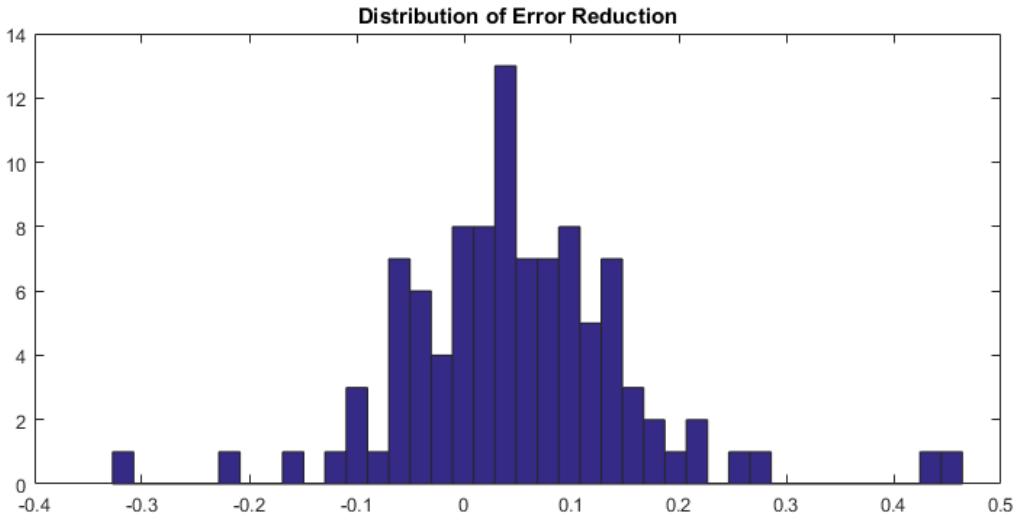


Figure 7.18: Histogram of Error Reduction

Since a positive number means that SOM does reduce test errors, statistically it turns out that 72% of the time SOM successfully decreases test errors made by standard RRBF, but in the rest 28% of the time, SOM undesirably increases the errors. The failure of SOM to decrease test errors can be partly due to a common setting of the learning rate η. Throughout the process only one single η is used. It is fixed during the training process, and over stocks. Therefore, if a changeable η is used, it is likely that SOM will remain effective more frequently. The following is a prediction comparison on a sample stock:

Electronic copy available at: https://ssrn.com/abstract=3307566

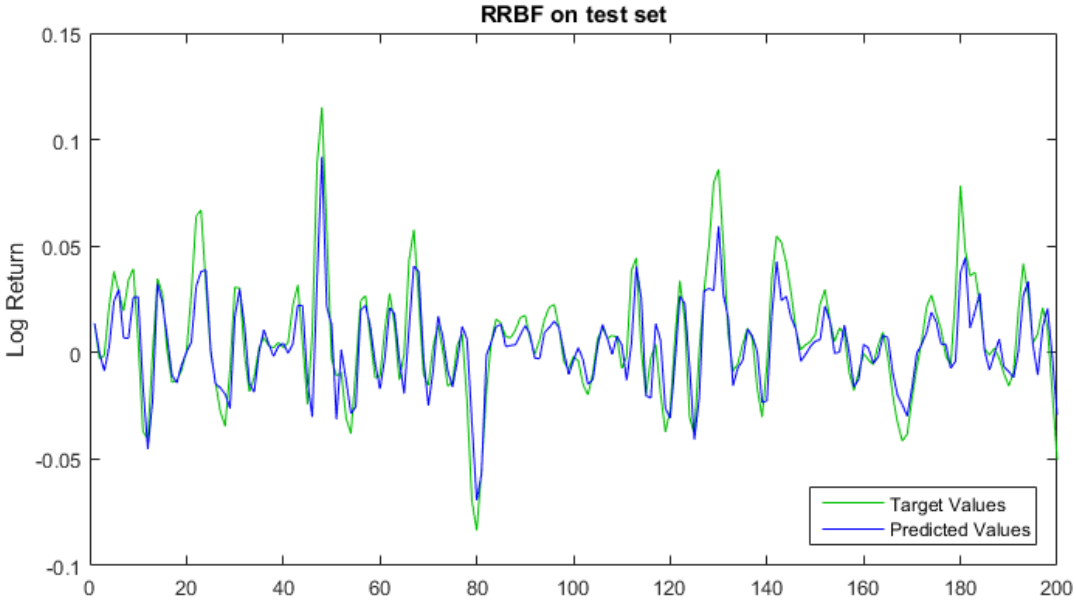


Figure 7.19: RRBF on test set

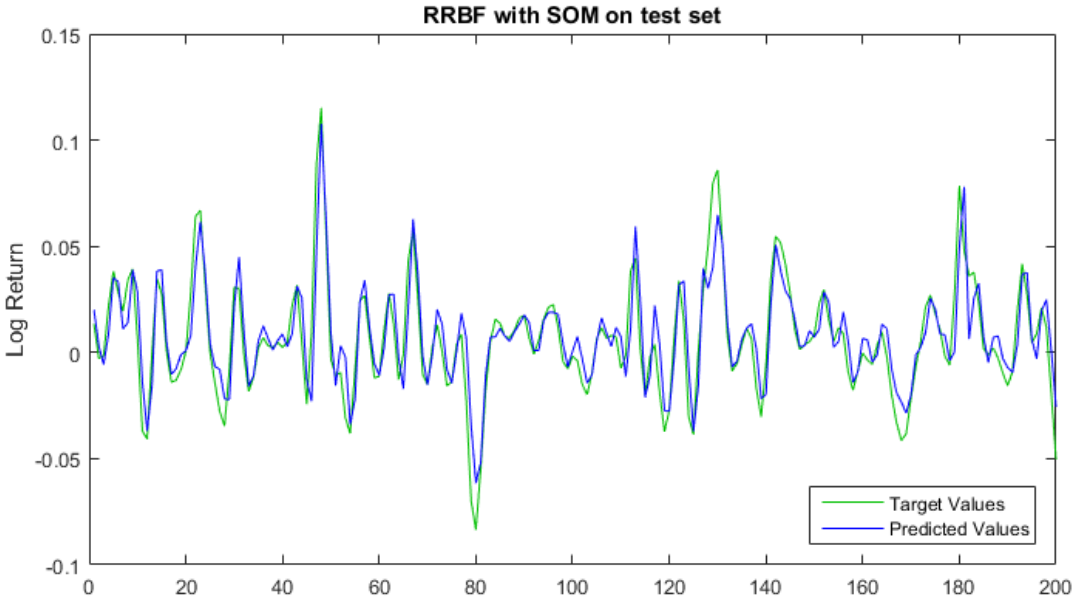


Figure 7.20: RRBF with SOM on test set

As for directional parametrised RRBF, the same experiments are done on the same 100 stocks using a common parameter setting. The results show that despite that on average training errors decrease by 6%, only 44% of the time

Electronic copy available at: https://ssrn.com/abstract=3307566

SOM manages to reduce the test error made by directional parametrised RRBF. Moreover, the training process takes much more time than previous experiments. The incompatibility of SOM to directional parametrised may be due to that directional parametrised RRBF itself has already scaled element-wise the distance between $\mathbf{U}(n)$ and $\mathbf{W}_{in}^{i}$, hence a further modification on $\mathbf{W}_{in}$ might create adverse effects.

Overall, the SOM model is promising and useful on standard RRBF in the sense that most of the time it is able to adaptively lead rows of $\mathbf{W}_{in}$ and $\mathbf{W}$ to the right positions when the original settings are not suitable for input dataset. Moreover, if the learning rate could be made flexible, or another $h(i, n)$ was utilised, chances are there for SOM to be further developed.

Electronic copy available at: https://ssrn.com/abstract=3307566

Part III

The mathematical formalism

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 8

Supervised learning

We have presented artificial and recurrent neural networks in Chapters (4) and (5), discussing the construction of neurons and introducing different networks with their respective algorithms. We are now going to give a more formal representation of such networks.

8.1 Introduction to learning theory

Vapnik [1982] developed a fundamental approach to learning theory called structural risk minimisation. We are going to introduce the main ideas.

8.1.1 Function estimation

8.1.1.1 Risk minimisation

In a general supervised learning problem we have two spaces of objects $\mathcal{X}$ and $\mathcal{Y}$ and would like to learn a function $h : \mathcal{X} \to \mathcal{Y}$ which outputs an object $y \in \mathcal{Y}$, given $x \in \mathcal{X}$. The function h is sometime called a score function or hypothesis. It is an element of some space of possible functions H called the hypothesis space. Given a sample $\{(x_1, y_1), ..., (x_n, y_n)\}$ where $x_i \in \mathcal{X}$ is an input and $y_i \in \mathcal{Y}$ is the corresponding response that we wish to get from $h(x_i)$. We consider probabilistic learning models, and assume that there is a joint probability distribution $P(X, Y)$ and that the sample is made of i.i.d variables drawn from that distribution. That is, we model uncertainty (such as noise in data) by letting y be a random variable with conditional distribution $P(y|x)$ for fixed x.
As stated by Vapnik [1992], the problem of learning is that of choosing from the given set of functions the one which approximates best the supervisor's response. A good approximation is usually defined with the help of a loss function. The loss function $L(\widehat{y}, y)$ is used to measure the inconsistency between the predicted value $\widehat{y}$ and the actual label y. It is a non-negative value, where the robustness of the model increases along with the decrease of the value of the loss function.
Thus, the goal of supervised learning is to choose a hypothesis h that minimise the expected loss (also called risk function)

$$\widehat{L}(h) = E[L(h(X), Y)] = \int L(h(X), Y) dP(X, Y) \tag{8.1.1}$$

where $L(\cdot, Y)$ is the loss function.

***Remark* 8.1.1** *This formulation implies that learning corresponds to the problem of function approximation. Thus, supervised learning can be seen as a function approximator.*

Electronic copy available at: https://ssrn.com/abstract=3307566

8.1.1.2 Empirical risk minimisation

In machine learning, given the samples $\{(x_i, y_i); i = 1, 2, ..., m\}$, of m pairs of input vector $x_i \in \mathbb{R}^{N_i}$ and labelled output vectors $y_i \in \mathbb{R}^{N_o}$, a task for supervised learning consists in learning a functional relation between the input x_i and the desired output y_i, for the training set. We can view supervised learning as a map from $\mathbb{R}^{N_i}$ to $\mathbb{R}^{N_o}$ where the information about the mapping is stored in a weight vector $W \in \mathcal{W}$. Given a map F, we train the network by adjusting the weights such that the resulting map is close to the desired map. Hence, the network is a distributed representation of the mapping.
The learning can be understood as finding a mapping $h(\cdot, w)$ such that $h(x_i, w) \approx y_i$ with w an N_w dimensional vector of parameters to be learned. An optimal scenario should allow for the algorithm to correctly determine the class labels for unseen instances. Thus, the goal of supervised learning is to choose a hypothesis h that minimise the expected loss in Equation (8.1.1). However, in most cases we do not know the joint probability distribution $P(X, Y)$ (see Definition (14.6.10)) to solve the expectation, and the only available information is contained in the training set. One solution is to consider the induction principle of Empirical Risk Minimisation (ERM) (see Vapnik [1992]):

- we substitute the sample mean for the expectation
- we find an hypothesis that minimise the empirical loss:

$$\overline{L}(h) = \frac{1}{m}\sum_{i=1}^{m} L(h(x_i), y_i)$$

  Also known as (A.K.A.) sample average approximation (see Appendix (14.5)).

We can write the loss function as $L(w; x_i, y_i)$ or $L(h(x_i, w), y_i)$. It is the hard core of empirical risk function and a significant component of structural risk function.
Under certain assumptions about the sequence of random variables $\{(x_i, y_i)\}$ (they are generated by a finite Markov process), if the set of hypotheses being considered is small enough, the minimiser of the empirical risk will closely approximate the minimiser of the expected risk as $m \to \infty$. Thus, for large enough m, we assume that the empirical loss approximates the expected value of the loss, $\overline{L} \approx \widehat{L}$. That is, we assume that the function $h(x, w_m^*)$ minimising $\overline{L}(w)$ over the set $w \in \mathcal{W}$ results in a risk (expected loss) $\widehat{L}(w_m^*)$ which is close to its minimum.

8.1.1.2.1 Convergence Formally, we need to assess if the risk $\widehat{L}(w_m^*)$ converges to its minimum value when $m \to \infty$ and find out the speed of convergence. Vapnik et al. [1989] showed it was equivalent to finding whether the empirical risk $\overline{L}(w)$ was converging uniformly to the actual risk $\widehat{L}(w)$ over the full set $h(x, w)$ for $w \in \mathcal{W}$, and what was the rate of convergence. Vapnik [1982] showed that bounds on the rate of uniform convergence were based on a quantitative measure of the capacity of the set of functions implemented by the learning machine: the VC-dimension of the set.
As an example, he considered the case of binary pattern recognition, for which $y \in \{0, 1\}$ and $h(x, w)$ is the class of indicator functions so that the loss function takes only two values $L(h(x, w), y) = 0$ if $y = h$ and it is equal to 1 otherwise. As a result, the risk functional is the probability of error, $P(w)$, and the empirical function, denoted $\nu(w)$ is the frequency of error. The VC-dimension of a set of indicator functions is the maximum number κ of vectors which can be shattered in all possible 2^κ ways using functions in the set. Bounds were obtained:

$$P(w) < \nu(w) + C_0(\frac{m}{\kappa}, \nu)$$

with confidence interval

$$C_0(\frac{m}{\kappa}, \nu) = \sqrt{\frac{\kappa\left(\ln\frac{2m}{\kappa} + 1\right) - \ln\nu}{m}}$$

which depends on the ratio $\frac{m}{\kappa}$.

Electronic copy available at: https://ssrn.com/abstract=3307566

8.1.1.2.2 Structural risk minimisation When the ratio $\frac{m}{\kappa}$ is large, the confidence interval gets small, and the actual risk is then bounded by only the empirical risk. However, if the ratio is small, the confidence interval cannot be neglected, and the minimisation of $P(w)$ requires a new principle, based on the simultaneous minimisation of $\nu(w)$ and the confidence interval. One must therefore control the VC-dimension of the learning machine. This is done by introducing a nested structure of subsets $S_p = \{h(x, w); w \in \mathcal{W}_p\}$ such that

$$S_1 \subset S_2 \subset \cdots \subset S_n$$

where corresponding VC-dimensions satisfy

$$\kappa_1 < \kappa_1 < \cdots < \kappa_n$$

The principle of structure risk minimization (SRM) requires: the empirical risk has to be minimised for each element of the structure. The optimal element S^* is then selected to minimise the guaranteed risk, defined as the sum of the empirical risk and the confidence interval. However, we must face a tradeoff: as κ increases the minimum empirical risk decreases, but the confidence interval increases.
Thus, SRM uses the VC-dimension as a controlling parameter for minimisation of generalisation bound. When designing statistical learning from data models, we can

- choose the appropriate structure and, keeping the confidence interval fixed, minimise the training error (empirical risk function). This is the approach in neural networks.
- keep the value of the training error fixed, and minimise the confidence interval. This is the approach in SVM.

8.1.1.2.3 Bias-variance tradeoff Considering issues with supervised learning, Geman et al. [1992] presented a tradeoff between bias and variance.

- Imagine that we have available several different, but equally good, training data sets. A learning algorithm is biased for a particular input x if, when trained on each of these data sets, it is systematically incorrect when predicting the correct output for x.
- A learning algorithm has high variance for a particular input x if it predicts different output values when trained on different training sets.

The prediction error of a learned classifier is related to the sum of the bias and the variance of the learning algorithm. A learning algorithm must be flexible enough (low bias) to fit the data, but not too flexible to avoid high variance. Some algorithms provide bias/variance parameter to adjust for this tradeoff. This is the case when adding a regularisation term to the loss function (SVM).

8.1.2 Solving the optimisation problem

8.1.2.1 The hypothesis function

In general, the relation between the input x_i and the desired output y_i can either be solved by a linear model as

$$\widehat{y}_i = W^\top x_i$$

where $W \in \mathbb{R}^{N_i \times N_o}$ is a weight matrix, or by a nonlinear model

$$\widehat{y}_i = f(W^\top x_i)$$

where $f(\cdot)$ is the activation function. There exists different ways of defining the activation function. Clevert et al. [2015] discussed how to find the most efficient ones in the case of deep learning.

Electronic copy available at: https://ssrn.com/abstract=3307566

As discussed previously, a neural network is a circuit composed of neurons representing a map between multiple inputs and a single output. As such it is a parallel computation device. To simplify exposition, we consider a simple two-layer network with input layer X, weights W and output Y. For each neuron Y_i, the behaviour of the output neuron is given by the transition probability

$$P(Y_i = y|W_i, X_i) = f_i(y, W_i, X_i) \tag{8.1.2}$$

where y is a scalar and f_i is the conditional probability mass function (see Definition (14.6.21)) or activation function. Note, as discussed in Remark (14.6.5), the probability is the conditional probability $P(Y_i = y|W_i = w, X_i = x)$. It is the probability for a neuron Y_i to respond by y when given a certain input with certain synaptic weights. The same type of equation can be given for multilayer network, but in that case X is the output of a previous layer.
Some examples of neurons appearing in neural network models are

- Linear non-linear (LN) neurons: These neurons first perform a linear action on their inputs, given by $h_i = W_i \cdot X_i$, and then a nonlinear function is applied to that value,

  $$P(Y_i = y|X_i, W_i) = f(y, h_i)$$

  Note, the output is random. In general, the activation function f is a sigmoidal function, h_i determine the probability of getting a certain output. Some examples of activation functions are

  - Exponential linear unit (ELU): The function is

    $$f(x) = \begin{cases} x \text{ if } x \geqslant 0 \\ \alpha(e^x - 1) \text{ otherwise} \end{cases}$$

  - Rectified linear unit (ReLU): The function is

    $$f(x) = \max(x, 0)$$

    The gradient is zero in the region where x is negative, and the neuron is inactive.
  - Leaky ReLU: The function is

    $$f(x) = \begin{cases} x \text{ if } x \geqslant 0 \\ \alpha x \text{ otherwise} \end{cases}$$

    It permits a weak gradient, when the neuron is inactive (small α).

- Bernoulli (binary) neurons: The neurons are binary, taking two values, typically 0 and 1. The probabilities are

  $$P(Y_i = 1|X_i, W_i) = f(1, h_i) \text{ and } P(Y_i = 0|X_i, W_i) = f(0, h_i) = 1 - f(1, h_i)$$

  We can equivalently say that it is the result of additive noise in the input. Denoting the noise by Z_i and $I(\cdot)$ the step function, we get

  $$\begin{aligned} Y_i &= I(h_i + Z_i) \\ P(Y_i = +1) &= \int_{z>-h_i}^{\infty} P(z)dz \end{aligned}$$

- Logistic neurons: They are binary neurons that behave according to the logistic equation

  $$P(Y_i = y) = f(h_i) = \frac{1}{1 + e^{-h_i}}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

8.1.2.2 The loss function

There exists a large number of loss function applied to neural networks (see Janocha et al. [2017] in the case of classification tasks). Some examples of loss function are:

- The $0-1$ loss function: $L(\widehat{y},y)=I(\widehat{y}\neq y)$, where $I(\cdot)$ is the indicator function.
- The mean square error (MSE) or quadratic loss function: $\frac{1}{m}\sum_{i=1}^{m}\big(y_i-h(x_i,w)\big)^2$ where $(y_i-\widehat{y}_i)$ is called the residual. It is widely used in linear regression. The method of minimising MSE is called ordinary least squares (OLS).
- L_2 norm: $\sum_{i=1}^{m}\big(y_i-h(x_i,w)\big)^2$. It is similar to the MSE but do not have the division by m.
- Other variants of the MSE exist such as the mean square logarithmic error where y_i is replaced by $\log(y_i+1)$. Further, one can consider other norms than the L_2 norm such as the L_1 norm, $\sum_{i=1}^{m}|y_i-h(x_i,w)|$.
- Maximum likelihood estimation: Choose probabilities to maximise the likelihood of the observed data

$$L(w;x_i,y_i)=-\log P(Y=y_i|X=x_i)$$

  An example of transition probability is the softmax function

$$P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}}$$

  where $s_{x_i}=h(x_i,w)$. In its simplest form, it can be used to measure the accuracy of a classifier. It is given by

$$\overline{L}(w;x,y)=-\frac{1}{m}\sum_{i=1}^{m}\log\widehat{y}_i$$

- The Poisson loss function: It measures how the predicted distribution diverges from the expected one. It is a variant of the Poisson distribution which is widely used for modelling count data. It is the limitting distribution for a normal approximation to a binomial one where the number of trials goes to infinity and the probability goes to zero. It is given by

$$\overline{L}(w;x,y)=-\frac{1}{m}\sum_{i=1}^{m}\big(\widehat{y}_i-y_i\cdot\log\widehat{y}_i\big)$$

- Hinge loss function for SVMs: It is a convex approximation to the $0-1$ loss function. It is generally used for maximum-margin classification, most notably for support vector machines (SVMs). The function is given by

$$\overline{L}(w;x,y)=\sum_{j\neq y_i}\big(s_j-s_{y_i}+1\big)^+$$

  where $s_{x_i}=h(x_i,w)$. In the special case of binary classification with intended output $y_i=\pm1$ and classifier $\widehat{y}_i$ the hinge loss is given by

$$\overline{L}(w;x,y)=\frac{1}{m}\sum_{i=1}^{m}\big(1-y_i\cdot\widehat{y}_i\big)^+$$

  When y_i and $\widehat{y}_i$ have the same sign (meaning $\widehat{y}_i$ predicts the right class) and $|\widehat{y}_i|>1$, the hinge loss equals to zero, but when they have opposite sign, it increases linearly with $\widehat{y}_i$ (one-sided error). A more general form is to

Electronic copy available at: https://ssrn.com/abstract=3307566

replace 1 with the margin ξ_m as a customised value. The more you violate the margin, the higher the penalty is. Since it is a one-sided error, it is not well-suited for regression-based problems. A variant of the hinge function is to square the residual, getting the squared hinge loss function.

- ϵ-insensitive loss (proposed by Vapnik): It is identical in behaviour to the absolute loss function, except that any points within some selected range ϵ incur no error at all. It is given by

$$\overline{L}(w;x,y) = \frac{1}{m}\sum_{i=1}^{m} R(y_i, \widehat{y}_i)$$

where

$$R(y,\widehat{y}) = \begin{cases} 0 \text{ if } |y-\widehat{y}| \leqslant \epsilon \\ |y-\widehat{y}| - \epsilon \text{ otherwise} \end{cases}$$

This is an error-free margin function which is ideal for SVM. With a suitable selection of ϵ we obtain sparsity of solutions.

One can also consider measures of distance. Some examples are

- Kullback-Leibler divergence: $D_{K,L}(P||Q) = \sum_y P(y)\log\frac{P(y)}{Q(y)}$. The relative entropy (information divergence/gain) is a measure of how one probability distribution diverges from a second expected probability distribution. The KL loss function is given by

$$\begin{aligned} \overline{L}(w;x,y) &= \frac{1}{m}\sum_{i=1}^{m} D_{K,L}(y_i||\widehat{y}_i) \\ &= \frac{1}{m}\sum_{i=1}^{m}\left[y_i \cdot \log\frac{y_i}{\widehat{y}_i}\right] \\ &= \frac{1}{m}\sum_{i=1}^{m}\left(y_i \cdot \log y_i\right) - \frac{1}{m}\sum_{i=1}^{m}\left(y_i \cdot \log \widehat{y}_i\right) \end{aligned}$$

where the first term is the entropy and the second term is called the cross-entropy. A KL number 0 indicates that we expect similar distributions, while 1 indicates that the distribution are totally different. Since it is a distribution-wise asymmetric measure, it does not qualify as a statistical metric.

- Cross entropy: $H(P,Q) = H(P) + D_{K,L}(P||Q)$ where $H(\cdot)$ is the entropy. It can be used in binary classification (labels are assumed to take values 0 or 1). The loss function is given by

$$\overline{L}(w;x,y) = \frac{1}{m}\sum_{i=1}^{m}\left[y_i \cdot \log\widehat{y}_i + (1-y_i)\cdot\log\left(1-\widehat{y}_i\right)\right]$$

It measures the divergence between two probability distribution. When CE is large the difference between the two distributions is large, and when CE is small the two distributions are similar.

8.1.2.3 Regularisation and stability

For the minimisation problem to have well-defined solution, we can place constraints on the set $\mathcal{H}$ of hypothesis being considered. One solution is to add a regularisation term to the loss function to avoid over-fitting on the training set (see bias-variance tradeoff):

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\overline{L}(w) = \frac{1}{m}\sum_{i=1}^{m} L(w; x_i, y_i)) + \lambda \cdot \Psi(w)$$

where λ is a regularisation parameter and $\Psi(\cdot)$ is a measure of distance. Some examples are:

- L_2 regularisation: $\Psi(w) = \|w\|^2 = \sum_{k,l} w_{k,l}^2$
- L_1 regularisation: $\Psi(w) = \sum_{k,l} |w_{k,l}|$
- elastic net regularisation: $\Psi(w) = \sum_{k,l} \beta w_{k,l}^2 + |w_{k,l}|$

These approaches are part of the Tikhonov regularisation. Some more complex regularisation methods are: Dropout, Batch normalisation, Stochastic depth, Fractional pooling, among others.

8.1.2.4 Solving for the optimum

Mathematically, the structural risk function of a model can be represented as

$$w^* = \arg\min_{w \in \mathcal{W}} \overline{L}(w) + \lambda \cdot \Psi(w) \tag{8.1.3}$$

One solution is to compute the gradient $\nabla_w \overline{L}(w)$, getting

$$\nabla_w \overline{L}(w) = \frac{1}{m}\sum_{i=1}^{m} \nabla_w L(w; x_i, y_i)) + \lambda \nabla_w \Psi(w)$$

When m is large it can be very expensive and one would rather consider the stochastic gradient descent (SGD), where the sum is approximated with minibatch of examples. See details below.

8.1.2.4.1 The batch gradient descent We can use batch gradient descent: we minimise the empirical loss, assuming it is convex and unconstrained. The gradient descent on the empirical loss is described in Algorithm (18).

Algorithm 18 Gradient Descent on the Empirical Loss

1: **repeat**
2: $w_{k+1} \leftarrow w_k - \eta_t \left(\frac{1}{m} \sum_{i=1}^{m} \frac{\partial L(w; x_i, y_i)}{\partial w} \right)$
3: **until** convergence or $k = K$

At each step, the gradient is the average of the gradient for all samples $i = 1, ..., m$. Thus, it is very slow when m is large.

8.1.2.4.2 The stochastic gradient descent Alternatively, we can compute the stochastic gradient descent (SGD) from just one sample (or a few samples). The algorithm is described in Algorithm (19).

Algorithm 19 Stochastic Gradient Descent on the Empirical Loss

1: **repeat**
2: $w_{k+1} \leftarrow w_k - \eta_t \frac{\partial L(w; x_i, y_i)}{\partial w}$
3: **until** convergence or $k = K$

Electronic copy available at: https://ssrn.com/abstract=3307566

Choose one sample i and compute the gradient for that sample only. Note, the gradient of one random sample is not the gradient of the objective function. However, the SGD converges to the empirical loss minimum as well as to the expected loss minimum. Nonetheless, convergence to the expected loss is slow

$$f(w_t) - E[f(w^*)] \leqslant O(\frac{1}{t}) \text{ or } O(\frac{1}{\sqrt{t}})$$

In principle, if the training set is small we consider the batch learning using quasi-Newton or conjugate gradient descent. On the other hand, if the training set is large we consider the stochastic gradient descent. However, the convergence is very sensitive to the learning rate. It needs to be determined by trial and error (model selection or cross-validation).

8.1.2.5 Online convex optimisation

Using the methods described in Section (10.2.2), Zinkevich [2003] proposed an algorithm for online convex optimisation. We let the feasible set $\chi \subseteq \mathbb{R}^n$ be a convex set. We have T convex cost functions $(c_1, c_2, ..., c_T)$, where each of the function is defined as $c_i : \chi \to [0, 1]$ See Algorithm (20).

Algorithm 20 Online Convex Algorithm

Require: : Import input vectors and corresponding labels.
1: Pick $x_1 \in \chi$ arbitrarily.
2: **for** $t = 1$ to T **do**
3: Step forward on the gradient direction: $\widehat{x}_{t+1} = x_t - \eta_t \nabla c_t(x_t)$
4: Project $\widehat{w}_{t+1}$ back to the feasible set: $x_{t+1} = Proj_\chi(\widehat{x}_{t+1})$
5: **end for**
6: **Return x**

Using the step size $\eta_t = \frac{1}{\sqrt{t}}$, the author proved that the regret of this online algorithm was bounded by

$$\sum_{t=1}^{T} c_t(x_t) - c_t(z_t) \leqslant \frac{D^2}{2}\sqrt{T} + G^2\sqrt{T} + 2D \cdot L(z_1, z_2, ..., z_T)\sqrt{T}$$

where $D = \max_{x,y\in\chi} \|x - y\|_2$ is the radius of the set, $\forall t$, $\forall x \in \chi$, $\|\nabla c_t(x)\|_2 \leqslant G$ is the upper bound of the gradient, and L is the total length of the drift, from z_1 to z_T given by $L(z_1, z_2, ..., z_T) = \sum_{i=1}^{T-1} \|z_{i+1} - z_i\|_2$.
As an example we consider the Hedge algorithm with n experts and construct a dimension for each expert, so that the feasible region lies in $\mathbb{R}^n$. Thus, we have

$$\chi = \{x : x_i \in [0, 1];\ \sum_{i=1}^{n} x_i = 1\}$$

Then, a feasible vector x encodes a distribution over experts, where exactly one expert is chosen, and expert i is chosen with probability x_i. We want to find the projection

$$Proj(y) = \arg\min_{x\in\chi} \|y - x\|_2$$

8.1.3 The dynamical system

We let the time-varying input signal be an N_ith order column vector $U(t) = [u_i(t)]$, the generated output is an N_oth order column vector $Y(t) = [y_o(t)]$ and the target output $\widehat{Y}(t)$ is an N_oth order column vector, where $t = 1, .., T$ and

Electronic copy available at: https://ssrn.com/abstract=3307566

T is the number of data points in the training set $\{(U(t), \widehat{Y}_t)\}$. As discussed in Section (8.1.1), the goal of supervised learning is to learn a function

$$Y(t) = f(\cdots, U(t-1), U(t))$$

such that $E(Y, \widehat{Y})$ is minimised, where E is an error measure.
The relation between the input $U(t)$ and the desired output $\widehat{Y}(t)$ can either be solved by a linear model as

$$Y(t) = W^\top U(t)$$

where $W \in \mathbb{R}^{N_i \times N_o}$, or by a nonlinear model.
In nonlinear models, we generally expand nonlinearly the input $U(t)$ into a high dimensional feature vector $X(t) \in \mathbb{R}^{N_x}$, and use linear models to get a reasonable output vector $Y(t)$. The output vector can be written as

$$Y(t) = f_{out}(W_{out}^\top X(t)) = f_{out}(W_{out}^\top \phi(\cdots, U(t-1), U(t)))$$

where $f_{out}(\cdot)$ is the output function (identity, sigmoid, or other), and $W_{out} \in \mathbb{R}^{N_x \times N_o}$ are the trained output weights. The functions

$$X(t) = \phi(\cdots, U(t-1), U(t))$$

transforming the current input $U(t)$ and its history $U(t-1), ...$ into a higher dimensional vector $X(t)$ are called kernels, and we refer to them in machine learning as expansion methods. Since these kernels have an unbounded number of parameters, we define them recursively as

$$X(t) = \phi(X(t-1), U(t)) \tag{8.1.4}$$

Expansion methods includes, among others, Support Vector Machines, Feedforward Neural Networks, Radial Basis Function approximators, Slow Feature Analysis, various Probability Mixture models.
As an example of such kernels, the recurrent neural networks (RNNs) can be written as

$$X(t+1) = f\big(W_{res}^\top \cdot X(t) + W_{in}^\top \cdot U(t+1) + W_{fb}^\top \cdot Y(t)\big) \tag{8.1.5}$$

where $f(\cdot)$ is an activation function, $W_{in} \in \mathbb{R}^{N_i \times N_x}$ is an input weight matrix, $W_{res} \in \mathbb{R}^{N_x \times N_x}$ is a connection weight matrix, and $W_{fb} \in \mathbb{R}^{N_o \times N_x}$ is a feedback weight matrix.
In addition to the function $f(\cdot)$, leaky integrator neurons performs a leaky integration of its activation from previous time steps, which can be applied before or after the activation function $f(\cdot)$. In the latter case, assuming no feedback, we get

$$X(t+1) = (1 - a\Delta t)X(t) + \Delta t f\big(W_{in}^\top \cdot U(t+1) + W_{res}^\top \cdot X(t)\big)$$

where Δt is a compound time gap between two consecutive time steps divided by the time constant of the system, and a is the delay (or leakage) rate. Setting $a = 1$ and redefining Δt as a constant α controlling the speed of the dynamics, we get

$$X(t+1) = (1 - \alpha)X(t) + \alpha f\big(W_{in}^\top \cdot U(t+1) + W_{res}^\top \cdot X(t)\big) \tag{8.1.6}$$

which is an exponential moving average. In general, we set $a\Delta t \in [0, 1]$ in the first equation and $\alpha \in [0, 1]$ in the second such that a neuron neither retain, nor leak, more activation than it had. Thus, with one extra parameter we make sure that neuron activations $X(t)$ never go outside the boundary defined by $f(\cdot)$. In fact, the neuron activation performs a low-pass filtering of its activations with the cutoff frequency

$$f_c = \frac{a}{2\pi(1-a)\Delta t}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

where Δt is the discretisation time step, allowing to tune the reservoirs for particular frequencies.
Non-temporal tasks using feedforward networks are functions, while RNNs may develop self-sustained temporal activation dynamics making them dynamical systems. Generally, supervised training of RNNs, such as gradient descent, adapt iteratively all weights according to their estimated gradients $\frac{\partial E}{\partial W}$ to minimise the output error $E = E(Y, \widehat{Y})$. One can adapt backpropagation (BP) methods from feedforward neural networks in the case of RNNs, by propagating the gradient through network connections and time. For example, the Backpropagation Through Time (BPTT) has a runtime complexity of $O(N_x^2)$ per weight update per time step for a single output ($N_o = 1$). Alternatively, the Real-Time Recurrent Learning (RTRL) estimate the gradients recurrently, forward in time, but it has a runtime complexity of $O(N_x^4)$. Improvements in standard RNNs design were proposed independently by Maass et al. [2002] under the name of Liquid State Machines and Jaeger [2001] under the name of Echo State Networks. Over time, these types of models became known as Reservoir Computing.

8.1.4 Handling memory

Maass et al. [2005a] [2005b] presented a computational theory characterising the gain in computational power that a fading memory system can acquire through feedback from trained readouts, in presence of noise. We briefly introduce this theory.
A map (or filter) F from input to output streams is defined to have fading memory if its current output at time t depends only on values of the input u during some finite time interval $[t - T, t]$.

Definition 8.1.1 *F has fading memory if there exists for every $\epsilon > 0$ some $\delta > 0$ and $T > 0$ so that*

$$|(Fu)(t) - (F\tilde{u})(t)| < \epsilon$$

for any $t \in \mathbb{R}$ and any input functions u, $\tilde{u}$ with $\|u(\tau) - \tilde{u}(\tau)\| < \delta$ for all $\tau \in [t - T, t]$.

This is a characteristic property of all filters that can be approximated by an integral over the input stream u, or more generally by Volterra or Wiener series.
In general, real-time computations involve memory or persistent internal states that can not be modelled with memory systems. It was shown that ANNs could be enlarged through feedback from trained readouts. As discussed above, ANNs are special cases of dynamical systems, which themselves can be seen as having universal capabilities for analog computing.

Theorem 8.1.1 *A large class $\mathcal{S}_n$ of systems of differential equations of the form*

$$x_i^{'}(t) = f_i\big(x_1(t), ..., x_n(t)\big) + g_i\big(x_1(t), ..., x_n(t)\big) \cdot v(t)\,, \; i = 1, ..., n \tag{8.1.7}$$

where $v(t)$ is the input vector, are universal for analog computing in the following sense:
It can respond to an external input $u(t)$ with the dynamics of any n-th order differential equation of the form

$$z^{(n)}(t) = G\big(z(t), z^{'}(t), z^{''}(t), ..., z^{(n-1)}(t)\big) + u(t) \tag{8.1.8}$$

for arbitrary smooth functions $G : \mathbb{R}^n \to \mathbb{R}$ if the input term $v(t)$ is replaced by a suitable memoryless feedback function $K\big(x_1(t), ..., x_n(t), u(t)\big)$ and if a suitable memoryless readout function $h\big(x_1(t), ..., x_n(t)\big)$ is applied to its internal state $\big(x_1(t), ..., x_n(t)\big)$.
Also, the dynamic responses of all systems consisting of several higher order differential equations of the form Equation (8.1.8) can be simulated by fixed systems of the form Equation (8.1.7) with a corresponding number of feedbacks.

The authors characterised Equation (8.1.7) as follows:

$$x_i^{'}(t) = -\lambda_i x_i(t) + \sigma\big(\sum_{j=1}^{n} a_{ij} \cdot x_j(t)\big) + b_i \cdot v(t)\,, \; i = 1, ..., n \tag{8.1.9}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

where $\sigma(\cdot)$ is a standard activation function. Note, it is similar to Equation (8.1.6). Further, if the activation function σ is also applied to the term $v(t)$, the system in Equation (8.1.9) can still simulate arbitrary differential equations of the form Equation (8.1.8) with bounded inputs $u(t)$ and bounded responses $z(t), ..., z^{(n-1)}(t)$.
As explained by Branicky [1995], all Turing machines can be simulated by systems of differential equations of the form Equation (8.1.8). Thus, systems of the form Equation (8.1.7), enlarged through feedback, are also universal for digital computing.
Casey [1996], Maass et al. [1998], among others, showed that additive noise, even with arbitrarily small bounded amplitude, reduces the non-fading memory capacity of any recurrent neural network to some finite number. As a result, arbitrary Turing machines can no-longer be simulated by such networks. Nonetheless, given this a priori limitation, feedback still provides noisy fading memory systems with maximum possible computational power. Thus, any finite state machine can be emulated by a fading memory system with feedback, despite the noise in the system.

Theorem 8.1.2 *Feedback allows linear and nonlinear fading memory systems, even in the presence of additive noise with bounded amplitude, to employ the computational capability and non-fading states of any given finite state machine for real-time processing of time varying inputs.*

Thus, the learning induced generation of high-dimensional attractors through feedback provides a new model for capturing the characteristics of persistent signal response.

8.2 SVM models

In machine learning, support vector machines (SVMs) are supervised learning models with associated learning algorithms that analyse data used for classification (see Vapnik [2000]) and regression analysis (see Drucker et al. [1996]). Introduced by Vapnik and Chervonenkis in 1963, Boser et al. [1992] suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes. SVM is a learning system using a high dimensional feature space where points are classified by means of assigning them to one of the two disjoint half spaces, either in the pattern space or in a higher dimensional feature space. The main objective is to identify maximum margin hyper plane, that is, the margin of separation between positive and negative examples is maximised. SVM finds the maximum margin hyper plane as the final decision boundary.

8.2.1 An optimisation problem

Assume that $x_i \in \mathbb{R}^d$, $i = 1, ..., N$ forms a set of input vectors with corresponding class labels (binary variables) $y \in \{+1, -1\}$, $i = 1, ..., N$. We want to find the maximum margin hyperplane that divides the group of points x_i for which $y_i = 1$ from the group of points for which $y_i = -1$, which is defined so that the distance between the hyperplane and the nearest point x_i from either group is maximised.
Any hyperplane can be written as the set of points x satisfying

$$w \cdot x - b = 0$$

where w is the normal vector to the hyperplane. The parameter

$$\frac{b}{\|w\|} \tag{8.2.10}$$

determines the offset of the hyperplane from the origin along the normal vector w. If the training data is linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the margin, and the maximum margin hyperplane is the hyperplane lying halfway between them. A linear classifier can be considered as a hyperplane with normal vector $w \in \mathbb{R}^d$ and offset b. The classification of x_i is determined by the hyperplane:

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\widehat{y}_i = \text{sgn}\big(w \cdot x_i + b\big)$$

where

$$\text{sgn}(x) = \begin{cases} -1 \text{ if } x < 0 \\ 0 \text{ if } x - 0 \\ 1 \text{ if } x > 0 \end{cases}$$

To prevent the data points from falling into the margin, we need to impose the following constraints: For each i either

$$w \cdot x_i - b \geqslant 1 \text{ if } y_i = 1$$

or

$$w \cdot x_i - b \leqslant -1 \text{ if } y_i = -1$$

That is, each data point must lie on the correct side of the margin. It can be rewritten as

$$y_i(w \cdot x_i - b) \geqslant 1 \, , \forall 1 \leqslant i \leqslant N$$

The objective of a linear classifier is to find the hyperplane that optimally separates the positive samples from negative ones. This is done by maximising the margin (gain) between the two regions. This as an optimisation problem where we maximise the margin M:

$$\max_{v,M} M$$

over the normal vector v parametrising the decision boundary such that $\|v\| = 1$ and $y \cdot v^\top x_i \geqslant M$. In that setting, the max-margin hyperplane is completely determined by those x_i which lie nearest to it. Hence, these x_i are called support vector.
We let $w = v \cdot M$ and make the substitution

$$\max_{w,M} M$$

such that $\|w\| = \frac{1}{M}$ and $\forall i : y_i \cdot w^\top x_i \geqslant 1$.
Given the inverse relationship between the norm $\|w\|$ and the margin M, maximising the distance between the planes is equivalent to minimising $\|w\|$. We can then transform this ratio into a minimisation problem

$$\min_w \|w\|^2$$

such that $\forall i : y_i \cdot w^\top x_i \geqslant 1$. It is a convex optimisation problem with constraints. However, this method only works if the data is linearly separable. Thus, to deal with non-separable data we need to introduce slack variables into each constraint: $\forall i : y_i \cdot w^\top x_i \geqslant 1 - \epsilon_i$, getting soft-margin SVM. To avoid the domination of the slack variables we introduce a penalty factor to the objective function, getting

$$\min_{w,\epsilon} \|w\|^2 + C \sum_i \epsilon_i$$

such that $\forall i : y_i \cdot w^\top x_i \geqslant 1 - \epsilon_i$ and $\epsilon_i \geqslant 0$ and $\lambda = \frac{1}{C}$. This this the primal offline SVM. It can be formulated with a new objective function to minimise the hinge loss as follows:

$$\min_w \|w\|^2 + C \sum_i \max\big(0, 1 - y_i w^\top x_i\big) \tag{8.2.11}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

where y_i is the ith target and $w \cdot x$ is the current output. It is a trade off where the first term keep the weights small, while the second term is a sum of hinge loss functions, which are high for a poor fit. Thus, computing the (soft-margin) SVM classifier amounts to maximising the margins, or, minimising the empirical risk function (see Section (8.1.2))

$$f(w,b) = \Big[\frac{1}{N}\sum_{i=1}^{N}\max\big(0, 1 - y_i(w \cdot x_i - b)\big)\Big] + \lambda\|w\|^2 \tag{8.2.12}$$

with hinge loss function

$$L(w; x_i, y_i) = \max\big(0, 1 - y_i(w \cdot x_i - b)\big)$$

The parameter λ determines the tradeoff between increasing the margin size and ensuring the the x_i lie on the correct side of the margin. So, for sufficiently small values of λ, the second term of the equation becomes negligible, and we recover the hard-margin SVM, if the data is linearly separable.
As discussed in Section (8.1.2), the SVP problem is equivalent to empirical risk minimisation (ERM) with Tikhonov regularisation, where the loss function is the hinge loss $L(x, y) = \max(0, xy)$. Thus, SVM is closely related to other fundamental classification algorithms, where the main difference is in the choice of the loss function.

8.2.2 The algorithms

In learning, we want to minimise that objective function. Most of the algorithms are iterative in nature and will hopefully converge to the optimal value. In order to find the break-even point between the algorithms we want to focus on the generalisation error, that is, the expected error on test set.
We can use online convex programming to optimise that problem, getting the Online SVM. Alternatively, we can consider offline algorithm and apply the Lagrange multiplier or the kernel trick.
In the former, from the representation property, at optimal solution, the weight vector w is given by $w = \sum_i \alpha_i y_i x_i$. In the latter, SVM can map the input vectors $x_i \in \mathbb{R}^d$ into a high dimensional feature space $\Phi(x_i) \in H$. A kernel function $k(x_i, x_j)$ performs the mapping $\phi(\cdot)$, that is, $k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$. The value w, in the transformed space, is given by $w = \sum_{i=1}^{N} \alpha_i y_i \phi(x_i)$. Dot products with w for classification can again be computed by the kernel trick as

$$w \cdot \phi(x) = \sum_{i=1}^{N} \alpha_i y_i k(x, x_i)$$

We are now present more formally some online and offline algorithms to solve the optimisation problem in Equation (8.2.11).

8.2.2.1 Online algorithms

8.2.2.1.1 A simple algorithm We can devise a simple online algorithm based on the gradient gradient descent (see Section (8.1.2.5)). Given the feasible set of the hyperplane $\mathcal{W} = \{w : \|w\|_2 \leqslant \lambda\}$ and the hinge loss function we get the online algorithm. See Algorithm (21) with learning rate $\eta_i = \frac{1}{\sqrt{i}}$ or $\eta_i = \frac{1}{i}$.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 21 SVM Online Algorithm

Require: : Import input vectors and corresponding labels.
1: Pick $w_1 \in \mathcal{W}$ arbitrarily.
2: **for** $i = 1$ to N **do**
3: The incurred loss is $L(w_i) \equiv hinge(w_i; x_i, y_i)$
4: Step forward on the gradient direction: $\widehat{w}_{i+1} = w_i - \eta_i \nabla L(w_i)$
5: Project $\widehat{w}_{i+1}$ back to the feasible set: $w_{i+1} = Proj_{\mathcal{W}}(\widehat{w}_{i+1})$
6: **end for**
7: **Return w**

Since the loss function is not differentiable we can use a sub-gradient to compute the gradient (see discussion below). We now briefly describe the projection (see Section (10.2.2)). Given the feasible set $\mathcal{W}$, the projection from $\widehat{w} \notin \mathcal{W}$ to its nearest point in $\mathcal{W}$ can be performed by multiplying $\widehat{w}$ with a scalar

$$w_{i+1} = Proj(\widehat{w}_{i+1}) = \frac{\lambda \cdot \widehat{w}_{i+1}}{\|\widehat{w}_{i+1}\|}$$

This a direct consequence from the parameter determining the offset hyperplane in Equation (8.2.10). In the case where $\widehat{w} \in \mathcal{W}$, then $Proj(\widehat{w}) = \widehat{w}$.

8.2.2.1.2 More complex algorithms Recent algorithms for finding the SVM classifier include sub-gradient descent (see Shalev-Shwartz et al. [2011]) and coordinate descent (see Hsieh et al. [2008]). Both techniques have proven to offer significant advantages over the traditional approach when dealing with large, sparse datasets. Sub-gradient methods are especially efficient when there are many training examples, and coordinate descent when the dimension of the feature space is high.
Sub-gradient descent algorithms for the SVM work directly with the expression $f(w, b)$ in Equation (8.2.12). Since f is a convex function, traditional gradient descent (or SGD) methods can be adapted, where instead of taking a step in the direction of the functions gradient, a step is taken in the direction of a vector selected from the function's sub-gradient (see Section (10.2.2))).
The coordinate descent algorithms apply to the dual problem. The coefficient α_i, $i = 1, ..., N$, is adjusted in the direction of $\frac{\partial L}{\partial \alpha_i}$. The resulting vector of coefficients $(\alpha_1^{'}, ..., \alpha_N^{'})$ is projected onto the nearest vector of coefficients that satisfies some Euclidean distances. The process is repeated until a near-optimal vector is obtained.

8.2.2.1.3 Parallel algorithm Zinkevich et al. [2009] proposed a parallel algorithm for Online SVM by introducing delayed updates. At the ith step the fetched gradient $\nabla L(w_{i-\tau})$ is the result at the τ-th previous step. They proved convergence. See Algorithm (22).

Algorithm 22 SVM Parallel Online Algorithm

Require: : Import input vectors and corresponding labels.
1: Pick $w_1 \in \mathcal{W}$ arbitrarily.
2: **for** $i = 1$ to N **do**
3: The incurred loss is $L(w_i) \equiv hinge(w_i; x_i, y_i)$
4: Step forward on the gradient direction: $\widehat{w}_{i+1} = w_i - \eta_i \nabla L(w_{i-\tau})$
5: Project $\widehat{w}_{i+1}$ back to the feasible set: $w_{i+1} = Proj_{\mathcal{W}}(\widehat{w}_{i+1})$
6: **end for**
7: **Return w**

Electronic copy available at: https://ssrn.com/abstract=3307566

8.2.2.2 Offline algorithms

8.2.2.2.1 The Lagrange multiplier As discussed in Section (10.4), the primal is a constrained optimisation problem with a differentiable objective function. By solving for the Lagrangian dual of the above problem, one obtains the simplified (unconstrained) optimisation problem called the dual problem. Since minimising $\|w\|^2$ is equivalent to minimising $\frac{1}{2}\|w\|^2$ we can introduce the Lagrange multipliers α_i. We can then write the objective function given in Equation (8.2.11) as

$$L(w,\alpha) = \frac{1}{2}\|w\|^2 - \sum_i^N \alpha_i\big(y_i w^\top x_i - 1\big)$$

where the new objective is

$$\min_w \max_\alpha L(w,\alpha)$$

We can then compute the KKT (Karush-Kuhn-Tucker) conditions for differentiable convex programs (see Section (10.4.1)). Applying the KKT theorem to the SVM optimisation problem we get the dual problem:

1. $\frac{\partial}{\partial w}L(w,\alpha) = 0 \rightarrow w = \sum_i \alpha_i y_i x_i$
2. $y_i w^\top x_i - 1 \geqslant 0$
3. $\sum_i \big(y_i w^\top x_i - 1\big) = 0$

From these conditions, we can see that

1. w can be represented as a linear combination of data points.
2. All data points are at least a normalised distance of 1 from the separating hyperplane.
3. As $\alpha_i \geqslant 0$ either $\alpha_i = 0$ or $y_i w^\top x_i = 1$ for all i.

Thus, the points for which $\alpha_i > 0$ are supporting the hyperplane, and as such are called support vectors. Hence, we write the set of support vectors as

$$\mathcal{S} = \{x_i : y_i w^\top x_i = 1\}$$

We can then substitute $w = \sum_i \alpha_i y_i x_i$ into the Lagrangian and get a simplified objective function

$$\begin{aligned} L(\alpha) &= \frac{1}{2}\big(\sum_i \alpha_i y_i x_i\big)\big(\sum_j \alpha_j y_j x_j\big) - \sum_i \alpha_i\Big[y_i\big(\sum_j \alpha_j y_j x_j\big)^\top x_i - 1\Big] \\ &= \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^\top x_j - \sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^\top x_j + \sum_i \alpha_i \\ &= \sum_i^N \alpha_i - \frac{1}{2}\sum_{i,j}^N \alpha_i\alpha_j y_i y_j x_i^\top x_j \end{aligned} \tag{8.2.13}$$

subject to $\sum_{i=1}^N \alpha_i y_i = 0$.
If we have α, we can then solve for w, and only need to solve for

$$\alpha^* = \arg\max_\alpha L(\alpha)$$

such that $\alpha_i \geqslant 0$, $\forall i$.

Electronic copy available at: https://ssrn.com/abstract=3307566

***Remark* 8.2.1** *The objective function only depends on inner products $x_i^\top x_j$, which is very useful for nonlinear classification tasks.*

This inner product can be replaced by a kernel function $k(x_i, x_j)$ which takes the inner product to a higher dimensional space. This is the basis of the kernel trick discussed below.

8.2.2.2.2 The kernel trick We can also apply the kernel trick. In some cases, for the data to be linearly separable we first need to apply a nonlinear transformation to a higher dimensional space. For instance, suppose that we would like to learn a nonlinear classification rule which corresponds to a linear classification rule for the transformed data points $\phi(x_i)$. However, when the dimension of $\phi(\cdot)$ becomes very large we should not use an explicit transformation. Note, if $\phi(x)$ is all ordered monomials of degree d, then $\phi^\top(x)\phi(x^{'}) = (x^\top x^{'})^d$ and we can implicitly work in a higher dimensional space by using a different dot product. This is called the kernel trick and

$$\phi^\top(x)\phi(x^{'}) = k(x, x^{'}) \tag{8.2.14}$$

is a kernel function (see Appendix (14.2.1.3)).
As a result of the representation property (see the dual problem above), at optimal solution, the weight vector w is a linear combination of the data points given by

$$w = \sum_i \alpha_i y_i x_i \tag{8.2.15}$$

Suppose we are given a kernel function defined as above. Then, the classification vector w, in the transformed space, satisfies

$$w = \sum_{i=1}^N \alpha_i y_i \phi(x_i)$$

Note that w can be an infinite dimensional vector (a function). We can treat the problem as a parameter estimation problem.
From Equation (8.2.14), the resulting decision boundary is

$$f(x) = \text{sgn}\big(w \cdot \phi(x) + b\big) = \text{sgn}\big(\sum_{i=1}^N y_i \alpha_i \cdot k(x, x_i) + b\big)$$

Given the dual problem in Equation (8.2.13), in order to estimate the values α_i, $i = 1, ..., N$ we solve the following quadratic programming problem (see Appendix (10.5))

$$\begin{aligned}
&\max_{\alpha_i}\Big(\sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{i=j}^N \alpha_i \alpha_j \cdot y_i y_j \cdot k(x_i, x_j)\Big) \\
&\text{subject to } 0 \leqslant \alpha_i \leqslant c \\
&\sum_{i=1}^N y_i \alpha_i = 0 \, , i = 1, ..., N
\end{aligned}$$

where the parameter c controls the tradeoff between the margin and misclassification error. In general it is set to $\frac{1}{2N\lambda}$. We can find some index i such that $0 < \alpha_i < c$ so that $\phi(x_i)$ lies on the boundary of the margin in the transformed space, and then solve

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} b = w \cdot \phi(x_i) - y_i &= \big(\sum_{k=1}^{N} \alpha_k y_k \phi(x_k) \cdot \phi(x_i)\big) - y_i \\ &= \big(\sum_{k=1}^{N} \alpha_k y_k k(x_k, x_i)\big) - y_i \end{aligned}$$

Then, new points can be classified by computing

$$z \to \text{sgn}(w \cdot \phi(z) - b) = \text{sgn}\Big(\big(\sum_{i=1}^{N} \alpha_i y_i k(x_i, z)\big) - b\Big)$$

We can also apply the same method to the primal optimisation problem. Replacing the weight vector w given in Equation (8.2.15) in the primal in Equation (8.2.11), the optimisation problem becomes equivalent to

$$\min_{\alpha} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^\top x_j + C \sum_i \max\big(0, 1 - y_i \sum_j \alpha_j y_j x_j^\top x_i\big)$$

Assuming that $w \in \text{Span}\{x_i, \forall i\}$, we replace $x_i^\top x_j$ with $k(x_i, x_j)$. We then let $\tilde{\alpha}_i = \alpha_i y_i$ (since we are not constrained we can flip signs arbitrarily) and obtain the objective function

$$\min_{\tilde{\alpha}} \frac{1}{2} \sum_{i,j} \tilde{\alpha}_i \tilde{\alpha}_j k(x_i, x_j) + C \sum_i \max\big(0, 1 - y_i \sum_j \tilde{\alpha}_j k(x_i, x_j)\big)$$

Using matrix notation, we rewrite the optimisation problem as

$$\min_{\tilde{\alpha}} \frac{1}{2} \tilde{\alpha}^\top K \tilde{\alpha} + C \sum_i \max\big(0, 1 - y_i f(x_i)\big)$$

where K is the Gram matrix, and $f(x) = f_\alpha(x) = \sum_j \tilde{\alpha}_j k(x, x_j)$.
Thus, in the case of polynomial or radial basis function, the parameters of the SVM model are (d, γ, c). In general, several levels are considered to determine these parameters efficiently.

8.2.3 Reproducing kernel Hilbert spaces

In some cases, rather than learning complicated nonlinear classifiers it is better to consider simpler, smoother functions. To do so, we will relate the formulation of the learning problem described in the kernel trick to a more familiar set up, that of Hilbert spaces, in order to understand its applicability.
We have already encounter the kernel function $k(x, \cdot)$ in the case of support vector machines (SVMs). In that setting, the problem can be expressed as

$$f^* = \arg\min_{f \in F} \big(\frac{1}{2}\|f\|^2 + \sum_{i=1}^{N} L(y_i, f(x_i))\big)$$

where $f = f_\alpha = \sum_i \alpha_i k(x_i, \cdot)$. Note, we will need to formally define F to reach our goal.

Electronic copy available at: https://ssrn.com/abstract=3307566

8.2.3.1 The Hilbert spaces

We let $\mathcal{H}_k$ be a reproducing kernel Hilbert space (RKHS), that is, $\mathcal{H}_k$ is a Hilbert space with some inner product $< \cdot, \cdot >$ and, for the set X, some positive definite kernel function $k : X \times X \to \mathbb{R}$ with the following pair of properties

$$\mathcal{H}_k = \{f : f = \sum_{i=1}^{\infty} \alpha_i k(x_i, \cdot)\}$$

It means that the space consists of all functions resulting from a linear combination of kernel evaluations:

$$< f, k(x_i, \cdot) >= f(x_i)$$

Hence, the kernel functions can be thought of as a kind of basis for the space. More formally, we have:

Definition 8.2.1 *RKHS*
A Hilbert space is called reproducing kernel Hilbert space for kernel function k if both of the following conditions are true:

1. *any function $f \in H$ can be written as an infinite linear combination of kernel evaluations*

$$f = \sum_{i=1}^{\infty} a_i k(x_i, \cdot)$$

   for $x_1, ..., x_m \in X$.

2. *$\mathcal{H}_k$ satisfies the reproducing property*

$$< f, k(x_i, \cdot) >= f(x_i)$$

   That is, the kernel function clamped to one x_i is the evaluating functional for that point.

This definition implies that

$$< k(x_i, \cdot), k(x_j, \cdot) >= k(x_i, x_j) \Leftarrow \text{ entries in the Gram matrix}$$

As a result, if we let F be an RKHS, $F = \mathcal{H}_k$, then we can rewrite the problem (empirical risk function) as

$$f^* = \arg\min_{f \in \mathcal{H}_k} \big(\frac{1}{2}\|f\|^2 + \sum_{i=1}^{N} L(y_i, f(x_i))\big)$$

As an example, we consider the square exponential kernel. We let $X \subset \mathbb{R}^N$ and

$$k(x, x^{'}) = e^{\frac{-\|x - x^{'}\|}{h}}$$

Evaluating this kernel at specific points results in Gaussian distributions. More interestingly we can also consider functions that are sums of Gaussian distributions. This is because the superposition of infinitely many Gaussian distributions is a dense set capable of approximating any continuous function.
We can now relate the empirical risk function, described above, to the kernel function as follows:

Theorem 8.2.1 *The representer theorem*
For any data set $\{(x_1, y_1), ..., (x_N, y_N)\}$, $\exists\ \alpha_1, ..., \alpha_N$ such that $f^* \in \arg\min\big(\frac{1}{2}\|f\|^2 + \sum_{i=1}^{N} L(y_i, f(x_i))\big)$ can be instead written as $f^* = \sum_{i=1}^{N} \alpha_i k(x_i, \cdot)$.

Electronic copy available at: https://ssrn.com/abstract=3307566

8.2.3.2 Nonparametric regression

The representer theorem is used in the case of nonparametric regression (NR): We want to learn some function $f : X \to \mathbb{R}$, but we do not have any prior knowledge about f, so that f can be an arbitrary function. One could use a loss function described above, but when the data contains many points in particular areas with large gaps of little or no data between the clusters, the resulting function could be very different from the target function in those areas. We therefore need to quantify how certain we are of the quality of the fit at various points on the function derived from regression on the data. We want to place confidence intervals around the function to reflect the areas of uncertainty. To do so we need to think in terms of fitting a distribution $P(f)$ over the target function f. We want the properties that low values of $\|f\|$ yield high values of $P(f)$ and vice versa. We can assume that the data (x_i, y_i) is such that $y_i = f(x_i) + \epsilon_i$, which is a function plus some error, the posterior $P(f|D)$. If we have the prior distribution $P(f)$, and the likelihood $P(y|f, x)$ we can compute the posterior distribution $P(f|y, x)$ via the application of Bayes' theorem

$$P(f|y, x) = \frac{P(f)P(y|f, x)}{P(y|x)}$$

We therefore need a suitable prior distribution $P(f)$ and a way to compute the posterior $P(f|D)$ where D is the domain. One solution is to use Gaussian processes as they are associated to the square exponential kernel and their distributions are easy to compute.

8.3 Forecasting models

8.3.1 Regression models

8.3.1.1 Data normalisation

Before training the network, the input data should be normalised [1] This process is usually important for complex machine learning models. Some models even require this procedure before data are fed into the models, since the models internally use distances or feature variances so that without normalisation the results would be heavily affected by the feature with the largest variance or scale. Normalising inputs could also help numerical optimisation method (such as Gradient Descent) converge much faster and more accurately.
There are different ways of normalising the data:

1. Linearly transform all the data to a certain range $[a, b]$, where the minimum of the dataset is mapped to a and the largest value is mapped to b.
2. Calculate the mean and standard deviation of the dataset, subtract each sample by sample mean, and then divide each sample by sample standard deviation.

In case [1], given a time series vector $\mathbf{X} = \{x_1, x_2, ..., x_N\}$, if the desired target range is in the interval $[b_L, b_H]$, with b_L and b_H being lower and upper bonds, then if $m_X = \max(\mathbf{X})$ and $n_X = \min(\mathbf{X})$ solve the equation

$$\begin{aligned} an_X + b &= b_L \\ am_X + b &= b_H \end{aligned}$$

we get the gradient a and intercept b, respectively:

$$\begin{aligned} a &= \frac{b_H - b_L}{m_X - n_X} \\ b &= b_L - an_X = \frac{b_L m_X - b_H n_X}{m_X - n_X} \end{aligned}$$

[1] Data normalisation is the process of transforming raw data into some standard form of data.

Electronic copy available at: https://ssrn.com/abstract=3307566

Thus $Y = aX + b$ will give the mapped data

$$\widehat{x}_i = (b_H - b_L)\frac{x_i - \min(X)}{\max(X) - \min(X)} + b_L \text{ for } i = 1, ..., N \tag{8.3.16}$$

where $\min(X)$ is the minimal value of all x_i, $\max(X)$ is the maximal value of all x_i, and $\widehat{x}_i$ is the normalised value of x_i, respectively. Note, we sometime need to perform operations on the normalised data within a sample window. When comparing the results among different windows we must always convert them back into the original order of magnitude using

$$x_i = [\max(X) - \min(X)]\frac{\widehat{x}_i - b_L}{(b_H - b_L)} + \min(X) \text{ for } i = 1, ..., N$$

In case [2], suppose $\mathbf{X}$ is a dataset, with $\bar{\mathbf{X}}$ being sample mean of the dataset and S being the sample standard deviation, then the normalised dataset $\mathbf{X}$ has the following expression:

$$\mathbf{X}' = \frac{\mathbf{X} - \bar{\mathbf{X}}}{S} \tag{8.3.17}$$

Sometimes we use the population mean μ instead of the sample mean $\bar{\mathbf{X}}$. For instance, one may assume the population mean of log return is zero. The choice varies from case to case generally.

In summary, data normalisation is an important part of the model which affects its prediction performance. The goal of normalisation is to make the data more tractable so that when the normalised data is further processed by the model, things become more controllable.

8.3.1.2 Description

In order to model returns with classic statistical models some statistical properties must be assumed such as stationarity or weak stationarity.
We let r_t be the log return of an asset at time t, and assume that $\{r_t\}$ is either serially uncorrelated or with minor lower order serial correlations, but it is dependent. We assume that r_t follows the dynamics

$$r_t = \mu_t + a_t \tag{8.3.18}$$

where a_t is the (stochastic) shock or mean-corrected return (or innovation) of an asset return [2]. The conditional mean and conditional variance of r_t, given the filtration $\mathcal{F}_{t-1}$, are defined by

$$\begin{aligned} \mu_t &= E[r_t|\mathcal{F}_{t-1}] = G(\mathcal{F}_{t-1}) \\ \sigma_t^2 &= Var(r_t|\mathcal{F}_{t-1}) = E[(r_t - \mu_t)^2|\mathcal{F}_{t-1}] = H(\mathcal{F}_{t-1}) \end{aligned}$$

where G and H are well defined functions with $H(\cdot) > 0$. The model for μ_t is the mean equation for r_t, and the model for σ_t^2 is the volatility equation for r_t.

***Remark* 8.3.1** *Some authors use h_t to denote the conditional variance of r_t, in which case the shock becomes $a_t = \sqrt{h_t}\epsilon_t$.*

[2] since $a_t = r_t - \mu_t$

Electronic copy available at: https://ssrn.com/abstract=3307566

Both the functions G and H could be non-linear. For simplicity of exposition we first consider the linear case. We will then briefly present some statistical models when H is non-linear. Since we assumed that the serial dependence of a stock return series was weak, if it exists at all, μ_t should be simple and we can assume that r_t follows a simple time series model such as a stationary $ARMA(p,q)$ model. In that case, the filtration is given by $\mathcal{F}_{t-1} = \{r_{t-1}, r_{t-2}, ..., r_{t-p}\}$ with p-lags and the drift satisfies

$$\mu_t = \phi_0 + \sum_{i=1}^{p} \phi_i r_{t-i} - \sum_{i=1}^{q} \theta_i a_{t-i} \tag{8.3.19}$$

where ϕ_i and θ_i are real numbers. The function H is constant, that is, $H(\mathcal{F}_{t-1}) = \sigma^2$. The parameters p and q are non-negative integers, and the order (p,q) of the ARMA model may depend on the frequency of the return series.

8.3.1.3 Linear regression

Linear regression (LR) is one of the simplest regression model which is still widely used in industry. This most straight forward method works as a benchmark or criterion when competed against a more complex model. The linear hypothesis for $h_\theta(\mathbf{X}^{(i)})$ is of the form

$$\theta_0 + \sum_{i=1}^{m} \theta_i \mathbf{X}^{(i)}$$

Typically, the root mean square error is calculated as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(h_\theta(\mathbf{X}^{(i)}) - \mathbf{y}^{(i)})^2}$$

For consistency with further discussions about recurrent neural network, $\mathbf{x} = \mathbf{X}^{(i)}$ is set to be the ith column instead of ith row of the design matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$. Then, assuming $\mathbf{y}$ is the target vector, in the sense of normal equation, the solution has the form

$$\hat{\boldsymbol{\theta}} = \mathbf{Y}\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}$$

In order to mitigate potential overfitting, a regularisation term β is introduced so that it becomes *ridge regression* and the solution turns into

$$\hat{\boldsymbol{\theta}} = \mathbf{Y}\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top + \beta\mathbf{I})^{-1} \tag{8.3.20}$$

where $\mathbf{I}$ is identity matrix. The following is a pseudo code for training linear regression:

Algorithm 23 LinearRegression Training

Function$linearTrain\mathbf{U}, \mathbf{Y}, modelInputs$ {$modelInputs \leftarrow N_u, N_y$ *etc.*}
$m \leftarrow nbPointsToIgnore + 1$;
$S = [\mathbf{1}; \mathbf{u}(:, m : end)]$;
$D = \mathbf{Y}(:, m : end)$;
$\mathbf{W}_{out} = DS^\top pinv(SS^\top + \beta I)$; {*Ridge Regression, 'pinv' is pseudo inverse*}
EndFunction

In the sections presenting some results, the task performance of linear regression is mainly used as a benchmark. If a complex model cannot surpass linear regression, then in terms of efficiency no one would use such complex model and would rather consider the simple linear regression.

Electronic copy available at: https://ssrn.com/abstract=3307566

8.3.2 The need to forecast volatility

8.3.2.1 Presentation

When visualising financial time series, one can observe heteroskedasticity [3], with periods of high volatility and periods of low volatility, corresponding to periods of high and low risks, respectively. We also observe returns having very high absolute value compared with their mean, suggesting fat tail distribution for returns, with large events having a larger probability to appear when compared to returns drawn from a Gaussian distribution. Hence, besides the return series, we must also consider the volatility process and the behaviour of extreme returns of an asset (the large positive or negative returns). The negative extreme returns are important in risk management, whereas positive extreme returns are critical to holding a short position. Volatility is important in risk management as it provides a simple approach to calculating the value at risk (VaR) of a financial position. Further, modelling the volatility of a time series can improve the efficiency in parameter estimation and the accuracy in interval forecast. As returns may vary substantially over time and appear in clusters, the volatility process is concerned with the evolution of conditional variance of the return over time. Users must make sure that volatilities and correlations are predictable and that their forecasts incorporate the most useful information available. As the forecasts are based on historical data, the estimators must be flexible enough to account for changing market conditions. One simple approach is to assume that returns are governed by the random walk model, and that the sample standard deviation $\hat{\sigma}_N$ or the sample variance $\hat{\sigma}_N^2$ of returns for N periods of time can be used as a simple forecast of volatility of returns, r_t, over the future period $[t+1, t+h]$ for some positive integer h. However, volatility has some specific characteristics such as

- volatility clusters: volatility may be high for certain time periods and low for other periods.
- continuity: volatility jumps are rare.
- mean-reversion: volatility does not diverge to infinity, it varies within some fixed range so that it is often stationary.
- volatility reacts differently to a big price increase or a big price drop.

These properties play an important role in the development of volatility models. As a result, there is a large literature on econometric models available for modelling the volatility of an asset return, called the conditional heteroscedastic (CH) models. Some univariate volatility models include the autoregressive conditional heteroscedastic (ARCH) model of Engle [1982], the generalised ARCH (GARCH) model of Bollerslev [1986], the exponential GARCH (EGARCH) of Nelson [1991], the stochastic volatility (SV) models and many more. Tsay [2002] discussed the advantages and weaknesses of each volatility model and showed some applications of the models. Unfortunately, stock volatility is not directly observable from returns as in the case of daily volatility where there is only one observation in a trading day. Even though one can use intraday data to estimate daily volatility, accuracy is difficult to obtain. The unobservability of volatility makes it difficult to evaluate the forecasting performance of CH models and heuristics must be developed to estimate volatility on small samples.

8.3.2.2 A first decomposition

As risk is mainly given by the probability of large negative returns in the forthcoming period, risk evaluation is closely related to time series forecasts. The desired quantity is a forecast for the probability distribution (pdf) $\tilde{p}(r)$ of the possible returns r over the risk horizon ΔT. This problem is generally decomposed into forecasts for the mean and variance of the return probability distribution

$$r_{\Delta T} = \mu_{\Delta T} + a_{\Delta T}$$

with

[3] Heteroskedastic means that a time series has a non-constant variance through time.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$a_{\Delta T} = \sigma_{\Delta T}\epsilon$$

where the return $r_{\Delta T}$ over the period ΔT is a random variable, $\mu_{\Delta T}$ is the forecast for the mean return, and $\sigma_{\Delta t}$ is the volatility forecast. The term $a_{\Delta T} = r_{\Delta T} - \mu_{\Delta T}$ is the mean-corrected asset return. The residual ϵ, which corresponds to the unpredictable part, is a random variable distributed according to a pdf $p_{\Delta T}(\epsilon)$. The standard assumption is to let $\epsilon(t)$ be an independent and identically distributed (iid) random variable. In general, a risk methodology will set the mean μ to zero and concentrate on σ and $p(\epsilon)$. To validate the methodology, we set

$$\epsilon = \frac{r - \mu}{\sigma}$$

compute the right hand side on historical data, and obtain a time series for the residual. We can then check that ϵ is independent and distributed according to $p(\epsilon)$. For instance, we can test that ϵ is uncorrelated, and that given a risk threshold α (say, 95%), the number of exceedance behaves as expected. However, when the horizon period ΔT increases, it becomes very difficult to perform back testing due to the lack of data. Alternatively, we can consider a process to model the returns with a time increment δt of one day, computing the forecasts using conditional averages. We can then relate daily data with forecasts at any time horizon, and the forecasts depend only on the process parameters, which are independent of ΔT and are consistent across risk horizon. The quality of the volatility forecasts is the major determinant factor for a risk methodology. The residuals can then be computed and their properties studied.

8.3.2.3 The structure of volatility models

8.3.2.3.1 Presentation The above heuristics being poor estimates of the future volatility, one must rely on proper volatility models such as the conditional heteroscedastic (CH) models. Since the early 80s, volatility clustering spawned a large literaturure on a new class of stochastic processes capturing the dependency of second moments in a phenomenological way. As the lognormal assumption is not consistent with all the properties of historical stock returns, Engle [1982] first introduced the autoregressive conditional heteroscedasticity model (ARCH) which has been generalised to GARCH by Bollerslev [1986].
We consider the log return r_t to follow the $ARMA(p,q)$ model described in the above Appendix. The excess kurtosis values, measuring deviation from the normality of the returns, are indicative of the long-tailed nature of the process. Hence, one can then compute and plot the autocorrelation functions for the returns process r_t as well as the autocorrelation functions for the squared returns r_t^2.

1. If the securities exhibit a significant positive autocorrelation at lag one and higher lags as well, then large (small) returns tend to be followed by large (small) returns of the same sign. That is, there are trends in the return series. This is evidence against the weakly efficient market hypothesis which asserts that all historical information is fully reflected in prices, implying that historical prices contain no information that could be used to earn a trading profit above that which could be attained with a naive buy-and-hold strategy which implies further that returns should be uncorrelated. In this case, the autocorrelation function would suggest that an autoregressive model should capture much of the behaviour of the returns.

2. The autocorrelation in the squared returns process would suggest that large (small) absolute returns tend to follow each other. That is, large (small) returns are followed by large (small) returns of unpredictable sign. It implies that the returns series exhibits volatility clustering where large (small) returns form clusters throughout the series. As a result, the variance of a return conditioned on past returns is a monotonic function of the past returns, and hence the conditional variance is heteroskedastic and should be properly modelled.

The conditional heteroscedastic (CH) models are capable of dealing with this conditional heteroskedasticity. The variance in the model described in Equation (8.3.18) becomes

$$\sigma_t^2 = Var(r_t|\mathcal{F}_{t-1}) = Var(a_t|\mathcal{F}_{t-1})$$

Electronic copy available at: https://ssrn.com/abstract=3307566

Since the way in which σ_t^2 evolves over time differentiate one volatility model from another, the CH models are concerned with the evolution of the volatility. Hence, modelling conditional heteroscdasticity (CH) amounts to augmenting a dynamic equation to a time series model to govern the time evolution of the conditional variance of the shock. We distinguish two types or groups of CH models, the first one using an exact function to govern the evolution of σ_t^2, and the second one using a stochastic equation to describe σ_t^2. For instance, the (G)ARCH model belongs to the former, and the stochastic volatility (SV) model belongs to the latter. In general, we estimate the conditional mean and variance equations jointly in empirical studies.

8.3.2.3.2 Benchmark volatility models The ARCH model, which is the first model providing a systematic framework for volatility modelling, states that

1. the mean-corrected asset return a_t is serially uncorrelated, but dependent
2. the dependence of a_t can be described by a simple quadratic function of its lagged values.

Specifically, setting $\mu_t = 0$ for simplicity, an $ARCH(p)$ model assumes that

$$r_t = h_t^{\frac{1}{2}} \epsilon_t \, , \, h_t = \alpha_0 + \alpha_1 r_{t-1}^2 + ... + \alpha_p r_{t-p}^2$$

where $\{\epsilon_t\}$ is a sequence of i.i.d. random variables with mean zero and variance 1, $\alpha_0 > 0$, and $\alpha_i \geqslant 0$ for $i > 0$. In practice, ϵ_t follows the standard normal or a standardised Student-t distribution. Generalising the ARCH model, the main idea behind (G)ARCH models is to consider asset returns as a mixture of normal distributions with the current variance being driven by a deterministic difference equation

$$r_t = h_t^{\frac{1}{2}} \epsilon_t \text{ with } \epsilon_t \sim N(0,1) \tag{8.3.21}$$

and

$$h_t = \alpha_0 + \sum_{i=1}^{p} \alpha_i r_{t-i}^2 + \sum_{j=1}^{q} \beta_j h_{t-j} \, , \, \alpha_0 > 0 \, , \, \alpha_i, \beta_j > 0 \tag{8.3.22}$$

where $\alpha_0 > 0$, $\alpha_i \geqslant 0$, $\beta_j \geqslant 0$, and $\sum_{i=1}^{\max(p,q)} (\alpha_i + \beta_i) < 1$. The latter constraint on $\alpha_i + \beta_i$ implies that the unconditional variance of r_t is finite, whereas its conditional variance h_t evolves over time. In general, empirical applications find the $GARCH(1,1)$ model with $p = q = 1$ to be sufficient to model financial time series

$$h_t = \alpha_0 + \alpha_1 r_{t-1}^2 + \beta_1 h_{t-1} \, , \, \alpha_0 > 0 \, , \, \alpha_1, \beta_1 > 0 \tag{8.3.23}$$

When estimated, the sum of the parameters $\alpha_1 + \beta_1$ turns out to be close to the non-stationary case, that is, mostly only a constraint on the parameters prevents them from exceeding 1 in their sum, which would lead to non-stationary behaviour. Different extensions of GARCH were developed in the literature with the objective of better capturing the financial stylised facts. Among them are the Exponential GARCH (EGARCH) model proposed by Nelson [1991] accounting for asymmetric behaviour of returns, the Threshold GARCH (TGARCH) model of Rabemananjara et al. [1993] taking into account the leverage effects, the regime switching GARCH (RS-GARCH) developed by Cai [1994], and the Integrated GARCH (IGARCH) introduced by Engle et al allowing for capturing high persistence observed in returns time series. Nelson [1990] showed that Ito diffusion or jump-diffusion processes could be obtained as a continuous time limit of discrete GARCH sequences. In order to capture stochastic shocks to the variance process, Taylor [1986] introduced the class of stochastic volatility (SV) models whose instantaneous variance is driven by

$$\ln(h_t) = k + \phi \ln(h_{t-1}) + \tau \xi_t \text{ with } \xi_t \sim N(0,1) \tag{8.3.24}$$

This approach has been refined and extended in many ways. The SV process is more flexible than the GARCH model, providing more mixing due to the co-existence of shocks to volatility and return innovations. However, one drawback

Electronic copy available at: https://ssrn.com/abstract=3307566

of the GARCH models and extension to Equation (8.3.24) is their implied exponential decay of the autocorrelations of measures of volatility which is in contrast to the very long autocorrelation discussed in Appendix (**??**). Both the GARCH and the baseline SV model are only characterised by short-term rather than long-term dependence. In order to capture long memory effects, the GARCH and SV models were expanded by allowing for an infinite number of lagged volatility terms instead of the limited number of lags present in Equations (8.3.22) and (8.3.24). To obtain a compact characterisation of the long memory feature, a fractional differencing operator was used in both extensions, leading to the fractionally integrated GARCH (FIGARCH) model introduced by Baillie et al. [1996], and the long-memory stochastic volatility model of Breidt et al. [1998]. As an intermediate approach, Dacorogna et al. [1998] proposed the heterogeneous ARCH (HARCH) model, considering returns at different time aggregation levels as determinants of the dynamic law governing current volatility. In this model, we need to replace Equations (8.3.22) with

$$h_t = c_0 + \sum_{j=1}^{n} c_j r^2_{t,t-\Delta t_y}$$

where $r_{t,t-\Delta t_y} = \ln(p_t) - \ln(p_{t-\Delta t_j})$ are returns computed over different frequencies. This model was motivated by the finding that volatility on fine time scales can be explained to a larger extend by coarse-grained volatility than vice versa (see Muller et al. [1997]). Thus, the right hand side of the above equation covers local volatility at various lower frequencies than the time step of the underlying data $(\Delta t_j = 2, 3, ..)$. Note, multifractal models have a closely related structure but model the hierarchy of volatility components in a multiplicative rather than additive format.

8.3.3 Generalised nonlinear nonparametric models

8.3.3.1 Presentation

Among the various techniques to forecast and classify financial time series, fundamental and technical analysis were the most popular ones. Even though statistical procedures were widely used for pattern recognition, the effectiveness of these methods relies both on model's assumptions and prior knowledge on data properties. To remedy these pitfalls, several classifiers developed, using various data mining and computational intelligence methods such as rule induction, fuzzy rule induction, decision trees, neural networks etc. For instance, the best recognised tools in the currency markets is the artificial neural networks (ANNs), supported by numerous empirical studies (see Ahmed et al. [2010]). Among the different networks existing, the artificial neural networks (ANNs) and the artificial recurrent neural networks (RNNs) are computational models designed by more or less detailed analogy with biological brain modules. The former is presented in Chapter (4) and the latter is introduced in Chapter (5). As discussed by Barron et al. [1988], neural networks can all be viewed as non-parametric methods for performing nonlinear regressions. This general class of methods is called Learning Networks. Thus, the foremost reason for using ANNs is that there is some nonlinear aspect to the forecasting problem under consideration, taking the form of a complex nonlinear relationship between the independent and dependent variables. The characteristics of financial time series, such as equity stock or currency markets, are influenced by the psychology of traders (behavioural finance) and are strongly nonlinear and hardly predictable (see Maknickiene et al. [2011]). Hence, neural networks are good candidates to learn the financial market.

8.3.3.2 Describing the models

Given a time series $\{x_t, x_{t-1}, ...\}$, we aim to learn from the known data to predict future values. That is, we want to estimate x at some future time

$$\widehat{x}_{t+h} = f(x_t, x_{t-1}, ..., x_{t-p}, z_1, z_2, .., z_q)$$

where h is the horizon prediction, and z_q is the qth other explanatory variable. Before training the network, the input data should be normalised into the interval $[b_L, b_H]$, with b_L and b_H being lower and upper bonds (see Section (8.3.1.1)). As discussed in Section (8.3.1), the statistical approach to forecasting time series involves the construction

Electronic copy available at: https://ssrn.com/abstract=3307566

of stochastic models to predict the value $\widehat{x}_{t+h}$ given previous observations. One approach is to use an $ARMA(p,q)$ model, but it lacks the ability to capture the nonlinear features of financial time series. Another approach is to extend the linear models to nonlinear ones. For example, we can generalise an $AR(p)$ model to become a *nonlinear autoregressive* (NAR) model (see Connor et al. [1994])

$$x_t = h(x_{t-1}, x_{t-2}, \cdots, x_{t-p}) + e_t, \tag{8.3.25}$$

where h is an unknown smooth function. Assuming that $E(e_t|x_{t-1}, x_{t-2}, \cdots) = 0$ and e_t has finite second moment, then the minimum mean square error optimal predictor of x_t with the knowledge of $x_{t-1}, ..., x_{t-p}$ is the conditional mean:

$$\widehat{x}_t = E(x_t|x_{t-1}, \cdots, x_{t-p}) = h(x_{t-1}, \cdots, x_{t-p}), t > p$$

Alternatively, as discussed in Section (8.1), we can use neural networks (NNs) (see Chapter (4)) to approximate the function and make some predictions.
Various authors showed that neural networks can be derived from the classical regularisation problem where some unknown function $y = h(x)$ can be approximated from the dataset $\{(x_t, y_t)\}$ and some smoothness constraints. In terms of regression analysis (see Section (8.3.1)) the function h is the conditional expectation of y_t given x_t. For instance, Lapedes [1987] showed that feedforward networks (see Section (4.1.2)) were NAR models for time series prediction. Poggio et al. [1990] followed a similar approach in the case of radial basis function (RBF).
For simplicity of exposition we consider the problem of mapping multiple input variables to a single output variable, like regression analysis. The predictor is given by the approximation of h as follow

$$\widehat{x}_t = \widehat{h}(x_{t-1}, \cdots, x_{t-p})$$

with

$$\widehat{h}(x_{t-1}, \cdots, x_{t-p}) = \sum_{i=1}^{N_h} w_i f(\sum_{j=1}^{p} w_{ij} x_{t-j} + b_i) \tag{8.3.26}$$

A simple way to visualise the structure of the network is to graph the connections between inputs, nonlinear hidden units and outputs. Figure 8.1 shows the general architecture of the NAR as an extension of the $AR(p)$ in a feedforward network. The weights, $\{w_i\}$ and $\{w_{ij}\}$ in Equation (8.3.26) can be seen as knobs defining the input-output function, $\widehat{h}$, of the network. N_h is the number of neurons in the second layer, f is a monotonic, smooth, and bounded function, b_i is a bias. The weights of the network, $\{w_i\}$ and $\{w_{ij}\}$ are estimated through supervised learning process given a time series sample $x_0, x_1, \cdots, x_n$, and the input-target pairs in the training set are $((x_{t-1}, x_{t-2}, \cdots, x_{t-p}), x_t)$, where $p > 0, t \geqslant p$.

More generally, to illustrate the nonlinearity of ANNs, we consider a multilayer network (see Section (4.2)) with $p = 1, .., P$ input-output training pairs where $p = P$ is the most recent observation, and assume a set of artificial neurons $\{f_{i,j,l}\}_{k=0}^{K}$ where the multilayer subscript $k = 0$ corresponds to the set of inputs $\{x_i\}_{i=1}^{N_0}$ and $k = K$ corresponds to the set of outputs $\{y_i\}_{i=1}^{N_K}$. For simplicity of exposition, we focus on a system with one hidden layer where the subscripts i and j represent the nodes on the input ($(K-2)$ layer) and the $(K-1)$th hidden layer, and the subscript s represent the nodes on the kth layer, respectively. In that setting, the output $O_{s,K}^{p}(x, w)$ can be expressed as

$$O_{s,K}^{p}(x,w) = g_{N_K}\Big(\sum_{j=1}^{N_{K-1}} g_{N_{K-1}}\big(\sum_{i=1}^{N_{K-2}} O_{i,K-2}^{p}(x,w) w_{i,j}^{k-2} + b_{j,K-1}\big) w_{j,s}^{k-1} + b_{s,K}\Big), s = 1, .., N_K \tag{8.3.27}$$

where $O_{i,K-2}^{p}(x,w) = x_{p,i}$ is the ith element of the vector input, and $g_{N_K}(\cdot)$ and $g_{N_{K-1}}(\cdot)$ are two possibly different functions. In the special case where $g_{N_K}(\cdot)$ is a linear function and $g_{N_{K-1}}(\cdot)$ is a sigmoid function, the system simplifies to

Electronic copy available at: https://ssrn.com/abstract=3307566

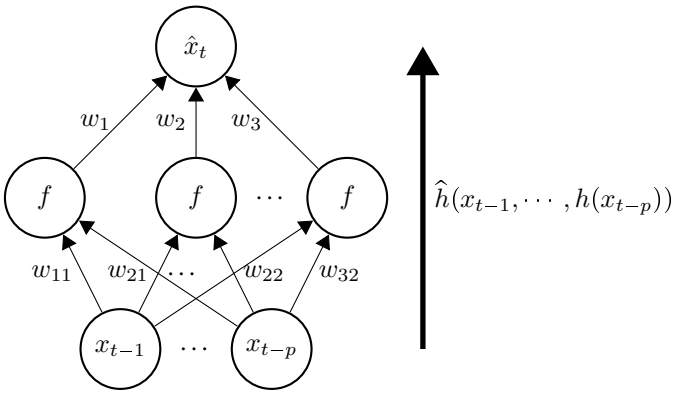


Figure 8.1: Feedforward network for nonlinear autoregressive model.

$$O^p_{s,K}(x,w) = \sum_{j=1}^{N_{K-1}} g_{n_{K-1}}\Big(\sum_{i=1}^{N_{K-2}} O^p_{i,K-2}(x,w) w^{k-2}_{i,j} + b_{j,K-1}\Big) w^{k-1}_{j,s} + b_{s,K} \ , s = 1, .., N_K$$

which can be compared to the ordinary least squares (OLS) regression

$$y^p_s = \alpha + \sum_{i=1}^{N_{K-2}} \beta_i x_{p,i} + \epsilon_s \ , s = 1, .., N_K$$

Hence, we see that the OLS regressor variables y^p_s are put through a transformation, or squashing function, given by $g_{N_{K-1}}(\cdot)$. By iterating within sample and examining the error terms, the neural network modify the input weights $\{w^{k-2}_{i,j}\}$ and the output weights $\{w^{k-1}_{j,s}\}$ to minimise within-sample measure for point prediction models. It can also maximise the classification rate of correct predictions for classification models.
It is not difficult to see that Equation (8.3.27) corresponds to the rate dynamics in Equation (8.3.18) with nonlinear G function. That is,

$$y = g_K\Big(a + \sum_{i=1}^{p} \phi_i x_i + \sum_{j=1}^{q} \theta_j g_{K-1}\big(\alpha_j + \sum_{i \to j} \omega_{ij} x_i\big)\Big)$$

where $x_1, ..., x_p$ are input values, y is the output, for the q-hidden nodes. If the activation function g_K is linear, the feedforward neural network (FFNN) becomes an autoregressive moving average model with nonlinear moving average part. Hence, the FFNN (with one hidden layer) is a generalisation of $ARMA(p,q)$ models.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 9

Reinforcement learning

See Kaelbling et al. [1996] for a survey on reinforcement learning, lectures from Li et al. [2017], Sompolinsky [2017], Levine et al. [2017] and textbooks by Sutton et al. [2017], Szepesvari [2010], among others.

9.1 Presentation

Reinforcement learning (RL) consists in solving problems involving an agent interacting with an environment, which provides numeric reward signals. The goal being to learn how to take actions in order to maximise reward. Some important terms regarding reinforcement learning are:

- Action a: All the possible moves that the agent can take.
- State s: Current situation returned by the environment.
- Reward r: An immediate return send back from the environment to evaluate the last action. It depends on the state s and the action a.
- Policy π: The strategy that the agent employs to determine next action based on the current state of the environment. It can be deterministic or stochastic.
- Value: The expected long-term return with discount, as opposed to the short-term reward.

We let $\mathcal{S}$ be the set of states and $\mathcal{A}$ is the set of actions.

Statement 9.1.1 *The sequence of events is as follows:*

$$s_t \to a_t \to (s_{t+1}, r_{t+1})$$

The actions change the state of the environment, which affects future movement possibilities. In a network model, the values are approximately:

- $s \sim X$ the input
- $a \sim Y$ the output

and the policy π reflects the calculation performed by the network to generate Y from X. In that sense it is a function from $\mathcal{S}$ to $\mathcal{A}$, $\pi : \mathcal{S} \to \mathcal{A}$. Further, the reward is a function from $\mathcal{S} \times \mathcal{A}$ to $\mathbb{R}$, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. At time step t and state s_t, the agent chooses an action a_t according to its policy $\pi(a_t|s_t)$, then the state and the environment changes to

Electronic copy available at: https://ssrn.com/abstract=3307566

new state s_{t+1} according to the dynamics $P(s_{t+1}|s_t, a_t)$ and the agent receives the reward $r_t = r(s_t, a_t)$. The process continues.
The goal of reinforcement learning is to maximise the expected cumulative reward

$$\pi^* = \arg\max_{\pi} E[\sum_{t\geqslant 0} r_t|\pi]$$

The RL anatomy is

$$\text{generate samples} \implies \text{fit a model / estimate return} \implies \text{improve the policy} \hookleftarrow$$

Similarly to supervised learning (SL), which is guided by error gradient signals, reinforcement learning is guided by feedback from the environment in the form of reward signals. However, some of the main differences are:

- Reward versus desired labels: There are no target labels in RL. Feedback is given in terms of some measures of success or failure, but the correct response is usually not provided. Since the system must work with partial information, we can not quantify how far we are from the right answer and as such there is no error gradient to compute.
- Dynamic versus static interaction with environment: SL (ANN) assumes a fixed stationary distribution of examples provided by the environment. However, in some real life problems the environment is dynamic in the sense that the next examples provided by the environment depends on the previous behaviour of the agent. Thus, we need to learn the desired trajectories rather than a fixed input output function. Since the statistics of the reward changes with time (it is path dependent) the actual reward may be delivered only later in the future, after a long sequence of actions.
- Look up tables versus distributed representations: Due to the inherent complexity of the task of the agent, the most common implementation of RL is in the form of a look-up table representations. It guarantees convergence of the algorithms for low dimensional problems. In the case of high dimension, implicit distributed representations such as neural networks are still under investigation.
- Markov decision process (MDP): Presently RL is performed in the theoretical framework of MDP. However, the underlying Markovian assumption is an important restriction, leaving out many real life situations.

Some examples of RL problems are:

- Robot locomotion: Objective: make the robot move forward; State: angle and position of the joints; Action: torques applied on the joints; Reward: 1 at each time step upright plus forward movement.
- Video games: Objective: complete the game with the highest score; State: raw pixel inputs of the game state; Action: game control (left, right, up, down); Reward: score increase/decrease at each time step.
- Go game: Objective: win the game; State: position of all pieces; Action: where to put the next piece down; Reward: 1 if win at the end of the game, 0 otherwise.
- Optimal portfolio selection: Objective: optimise the wealth of an agent at some future time; State: stock price; Action: choose between between two different investments, a risky asset and a risk-free one; Reward: payoff at each time step. See details in Appendix (14.8).
- Option pricing: it can be cast into an optimal portfolio selection problem (see Section (13.1)).

9.2 Learning with stationary reward

We consider a Markov reward process (MRP) defined by the tuple $(\mathcal{S}, \mathcal{R}, \mathbb{P}, \gamma)$ where $\mathcal{S}$ is the set of possible states, $\mathcal{R}$ is the distribution of reward given $(state, action)$ pair, $\mathbb{P}$ is the transition probability, γ is the discount factor.

Electronic copy available at: https://ssrn.com/abstract=3307566

9.2.1 Description

To account for the lack of explicit gradient signals (no target labels) one solution is to use neural networks (NNs) with built-in stochasticity. This noise will be used to form exploration of the state space of $\mathcal{S}$ such that an estimate of the gradient is implicitly evaluated, in a process similar to the stochastic gradient descent discussed in the context of online learning.
In Section (8.1.1), we have considered a simple two-layer network with input layer X, weights W and output Y. So far, we have assumed fixed parameters W_i and some input X_i. However, in reinforcement learning, after performing an action, the system receives a reward. It then updates its weights accordingly, with the desire to maximise future rewards. Thus, we let the policy be represented by a network where the input represent the state, the output is the action selection probabilities, and the weights are the policy parameters. We let W be the vector of parameters and ρ is the performance of the corresponding policy, that is, the average reward per step.
At each time step we do:

1. The environment provides an input X_{input} to the network according to some probability distribution $P(X_{input})$.
2. The network use the transition probabilities to generate the outputs $\{Y_i\}$ (can be the activities of all neurons in the network) in a feedforward manner corresponding to the action taken by the system.
3. Based on the output, the environment creates the reward signal which is a scalar r. The environment is generally stochastic, so that there is a probability distribution for the reward given the outputs, that is, $P(r|\{Y_i\})$.
4. The reward signal is a global scalar broadcasted to the entire network. All connections W_{ij} are updated in a way that will maximise the average reward. That is, maximising $\rho =< r >= R(W)$ [1], where the average is with respect to
   (a) The inputs X_{input}, according to $P(X_{input})$.
   (b) The conditional distribution of the outputs Y_i, according to $g_i(y, W_i, X_i)$.
   (c) The conditional rewards distribution r_i, denoted $P(r|\{Y_i\}, X_{input})$.

It is generally assumed that all of these external distributions are stationary, that is, they are independent of time. The goal is to find

$$\arg\max_W < r >$$

over all the parameters of the network. We want to find the set of weights that guarantees the highest possible rewards, over enough time. One way of achieving such a result is to use the REINFORCE algorithm.

9.2.2 The REINFORCE algorithm

In this algorithm (see Williams [1992]) (Derivation is given in Section (9.4.3)) the weights are updated according to the following rule:

$$\Delta W_{ij} = \alpha_{ij}\big(r - b_{ij}\big)e_{ij}\ ,\ e_{ij} = \frac{\partial \ln g_i}{\partial W_{ij}}$$

where α_{ij} is the learning rate and g_i is the the conditional probability mass function given in Equation (8.1.2). This value is independent of the reward and the output. r is the instantaneous global reward signal. b_{ij} is the reward threshold, also known as the baseline, and it is independent from the reward and the output. e_{ij} is the eligibility trace representing the responsibility of a particular connection to the generation of the output Y_i, and is therefore a stochastic

[1] Given two random variables X, Y in L_2, the inner product is $< X, Y >= E[XY]$. We denote $< X >= E[X]$ the expectation of X. In the case where $X = (X_1, ..., X_n)$ we can write $< a, X >= \sum_{i=1}^n a_j X_j$.

Electronic copy available at: https://ssrn.com/abstract=3307566

quantity depending on the input and the network activities, but not on the reward.
The claim of the algorithm states: for a given vector W, we have

$$\sum_{i,j} < \Delta W_{ij} > \frac{\partial R(W)}{\partial W_{ij}} > 0$$

where $< r >= R(W)$ is a function of W. That is, on average, the change in weights has a positive projection on the reward gradient.
One can prove that if $\alpha_{ij} = \alpha$ (a constant) then

$$< \Delta W_{ij} > \propto \frac{\partial < r >}{\partial W_{ij}}$$

implying that we are moving in the exact direction of the gradient.
Proof: Given the policy $g_i = P(Y_i = y|W_i, X_i)$ (see Equation (8.1.2)), from the normalisation of the mass function (see Lemma (14.6.2) for the pmf), we have

$$\sum_{y_i} g_i(y_i, W_i, X_{input}) = 1$$

From the chain rule, and since r is independent from W, we have

$$\frac{\partial \ln g_i}{\partial W_{ij}} = \frac{1}{g_i(y_i, W_i, X_i)} \frac{\partial g_i}{\partial W_{ij}}$$

The eligibility trace is a random variable with mean zero (over y_i),

$$< e_{ij} >=< \frac{1}{P(Y_i = y_i|W_i, X_i)} \frac{\partial g_i}{\partial W_{ij}} > \sum_{y_i} \frac{\partial g_i}{\partial W_{ij}} = 0$$

From Theorem (14.6.3), we have

$$\begin{aligned}
< \Delta W_{ij} > &= \sum_{r, X_{input}, y_i} \alpha_{ij}(r - b_{ij}) \frac{1}{g_i(y_i, W_i, X_i)} \frac{\partial g_i}{\partial W_{ij}} \left[P(X_{input})P(Y_i = y_i|W_i, X_i)P(r|\{Y_i\}, X_{input})\right] \\
&= \alpha_{ij} \sum_{r, X_{input}, y_i} P(X_{input})P(r|\{Y_i\}, X_{input}) r \frac{\partial g_i}{\partial W_{ij}} - \alpha_{ij} \sum_{X_{input}} P(X_{input}) b_{ij} \frac{\partial g_i}{\partial W_{ij}} \\
&= \alpha_{ij} \frac{\partial}{\partial W_{ij}} \sum_{r, X_{input}, y_i} P(X_{input})P(r|\{Y_i\}, X_{input}) g_i(y_i, W_i, X_{input}) \\
&= \alpha_{ij} \frac{\partial}{\partial W_{ij}} R(W)
\end{aligned}$$

where we have used the fact that $P(r|\{Y_i\}, X_{input})$ is independent of W to pull the gradient with respect to W outside of the sums.
The result is

$$\sum_{i,j} < \Delta W_{ij} > \frac{\partial R(W)}{\partial W_{ij}} = \sum_{i,j} \alpha_{ij} \left(\frac{\partial R(W)}{\partial W_{ij}}\right)^2$$

which proves the claim.
Thus, for $\alpha_{ij} = \alpha$ the algorithm is a stochastic ascent process. For a given weight W, we get

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}\Delta W &= <\Delta W> +\eta \\ &= \alpha\nabla_W R(W) + \eta\end{aligned}$$

so that η is a zero mean noise ($<\eta>=0$). As a result, all the theorems regarding convergence of STA theory apply to this algorithm.

9.2.2.1 Some examples

9.2.2.1.1 Bernoulli neurons In the case of Bernoulli elements we have $g_i(Y_i=1)=P_i$ and $g_i(Y_i=0)=1-P_i$. Assuming a neuron with no input (a neuron whose input is always 1), we want to learn the value of P_i maximising the reward (Two arm bandit problem). We can compute the eligibility trace as

$$\begin{aligned}e_{ij} &= \frac{\partial \log g_i}{\partial P_i} \\ &= \frac{1}{g_i}\frac{\partial g_i}{\partial P_i} \\ &= \frac{1}{P_i(1-P_i)}(Y_i - P_i)\end{aligned}$$

which is dependent on the value of Y_i. We can also compute its expectation as

$$\begin{aligned}<e_{ij}> &= \frac{1}{P_i(1-P_i)} <Y_i - P_i> \\ &= \frac{1}{P_i(1-P_i)} <Y_i - <Y_i>> \\ &= \frac{1}{<\delta Y_i^2>} <Y_i - <Y_i>>\end{aligned}$$

Thus, the expected value of the eligibility trace is zero and its variance is $<\delta Y_i^2>$.
If we choose $\alpha_{ij}=\eta_i P_i(1-P_i)$ and $b_i=0$, the learning rule simplifies to

$$\Delta P_i = \eta_i r(Y_i - P_i)$$

Assuming $r>0$, it will converge to $P_i=1$ if $<r|1>><r|0>$ and vice versa.

9.2.2.1.2 Binary LN neurons In the case of a binary LN neuron we have

$$\begin{aligned}P_i &= g_i(1, W_i, X_i) \\ &= \frac{1}{1+e^{-W_i\cdot X_i}}\end{aligned}$$

so that P_i is a function $h_i \equiv W_i \cdot X_i$. We get

$$\begin{aligned}\frac{\partial P_i}{\partial W_{ij}} &= \frac{X_{ij}e^{-W_i\cdot X_i}}{\left(1+e^{-W_i\cdot X_i}\right)^2} \\ &= P_i(1-P_i)X_{ij}\end{aligned}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

Since we are varying W, we need to calculate the eligibility trace

$$\begin{aligned} e_{ij} &= \frac{\partial \log g_i}{\partial P_i} \cdot \frac{\partial P_i}{\partial W_{ij}} \\ &= \frac{1}{P_i(1-P_i)}(Y_i - P_i)P_i(1-P_i)X_{ij} \\ &= (Y_i - P_i)X_{ij} \end{aligned}$$

If we choose $\alpha_{ij} = \eta > 0$ and $b_{ij} = 0$, the earning rate simplifies to

$$\Delta W_{ij} = \eta r(Y_i - P_i)X_{ij}$$

The form $(Y_i - P_i)X_{ij}$ is the classic Hebbian unsupervised learning.
To guarantee convergence we need some bounds on the fluctuation of the reward, which can be sensitive to the choice of the baseline b_{ij}. It can make the convergence of the learning algorithm very slow so that one we be careful when choosing the baseline. A popular choice is

$$b_{ij} = \overline{r}(t)$$

where

$$\overline{r}(t) = r(t-1)(1-\gamma) + \gamma \overline{r}(t-1)\,, 0 < \gamma < 1$$

It is a sort of integration of all of the rewards up to the current point in time, where the rewards from the past diminish exponentially. Thus b_{ij} is an estimate of the expected reward based on past history. It is a temporary average of the reward where γ is a decay parameter controlling how far back in time the system looks. It diminishes the rewards from the far past to nothing, while keeping the rewards from the near past relevant.

9.3 Markov decision process

In the case where the reward is a stochastic process the problem becomes that of a stochastic control problem (see Appendix (14.7)).
One can use Markov decision processes (MDP) (see Puterman [1994]) to formulate the RL problem. An MDP is a Markov reward process (MRP) with decision where $\mathcal{A}$ is a finite set of actions. The current state completely characterise the state of the world. The MRP is defined by the tuple $(\mathcal{S}, \mathcal{R}, \mathbb{P}, \gamma)$ while the MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$ where $\mathcal{S}$ is the set of possible states, $\mathcal{A}$ is the set of possible actions, $\mathcal{R}$ is the distribution of reward given $(state, action)$ pair, $\mathbb{P}$ is the transition probability, γ is the discount factor.
There exists several types of MDP problems which we classifies as follows:

1. Discrete time
   - countably finite state and action spaces
   - countably infinite state and/or action spaces
2. Continuous time
   - requires partial differential equations, the Hamilton-Jacobi-Bellman (HJB) equation
   - limiting case of Bellman equation as time-step tends to zero.

We are going to present some problems of interest to reinforcement learning associated to the case of countably finite state and action spaces. The continuous time MDP is presented in Appendix (14.7).

Electronic copy available at: https://ssrn.com/abstract=3307566

9.3.1 The problem

In general, the dynamics of the environment depends on the entire history of the system (all the previous states and actions). In that case, the transition probability is given by

$$P\big(s_{t+1} = s^{'}, r_{t+1} = r^{'}|(s_t, a_t), (s_{t-1}, a_{t-1}), ..., (s_0, a_0)\big)$$

In the case of the Markov decision process (MDP), the current change in state and the reinforcement received only depend on the previous state and action. This is the Markov property (see Appendix (14.6.8)). A Markov chain is a stochastic process with the Markov property (see Definition (14.6.26)) which is written as

$$P\big(s_{t+1} = s^{'}, r_{t+1} = r^{'}|(s_t, a_t), (s_{t-1}, a_{t-1}), ..., (s_0, a_0)\big) = P\big(s_{t+1} = s^{'}, r_{t+1} = r^{'}|(s_t, a_t)\big)$$

if both conditional probabilities are well defined. As discussed in Remark (14.6.6), Markov chains can be described by a sequence of directed graphs, where the edges of graph t are labelled by the probabilities of going from one state at time t to the other states at time $t + 1$, denoted $P(X_{t+1} = x|X_t = x_t)$. The latter is the probability of the state of the machine, which describes the statistical behaviour of the machine with an element x_t of the state space as input.
We assume that the state s is chosen from a finite set $\mathcal{S}$ and, similarly, the actions a is chosen from a finite set $\mathcal{A}$. Given the sequential order in Statement (9.1.1), we can define a policy as follows:

Definition 9.3.1 *A policy π is the probability of an action a given the state s*

$$\pi(s, a) = P(a_t = a|s_t = s) = P(a|s)$$

Note, we will sometime denote $\pi(s_t, a_t)$ to represent the probability $P(a_t|s_t)$ when focusing on the variables (s_t, a_t). Further, to emphasise a parametric policy we will denote $\pi_\theta(s, a) = P(a_t = a|s_t = s, \theta)$ where $\theta \in \mathbb{R}^l$, for $l << |\mathcal{S}|$, is a parameter vector.
A policy fully defines the behaviour of an agent. MDP policies depend on the current state and not the history. Thus, they are stationary:

$$a_t \sim \pi(\cdot|s_t)\,, \forall t > 0$$

In the case where it is a deterministic function, it can behaves like a δ-function (there is a probability 1 to take an action $a^{'}$ given a state s, and a probability 0 to take all other actions).

9.3.1.1 The transition probabilities

In Markov chain (see Appendix (14.6.8)), we let $\mu_{t,i} = P(S_t = i)$ be a vector of probabilities, the mass function of s_t, and define the transition operator $\mathcal{T}$ as

$$\mathcal{T}_{i,j} = P(s_{t+1} = i|s_j = j)$$

so that $\mu_{t+1} = \mathcal{T}\mu_t$. We denote the transition probability as $P(s_{t+1}|s_t)$. Accounting for the policy, we let $\xi_{t,k} = P(a_t = k)$ and write the state-action transition operator as

$$\mathcal{T}_{i,j,k} = P(s_{t+1} = i|s_j = j, a_t = k)$$

so that

$$\mu_{t+1,i} = \sum_{j,k} \mathcal{T}_{i,j,k}\mu_{t,j}\xi_{t,k}$$

We get the relation

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{bmatrix} s_{t+1} \\ a_{t+1} \end{bmatrix} = \mathcal{T} \begin{bmatrix} s_t \\ a_t \end{bmatrix} \text{ and } \begin{bmatrix} s_{t+k} \\ a_{t+k} \end{bmatrix} = \mathcal{T}^k \begin{bmatrix} s_t \\ a_t \end{bmatrix}$$

For simplicity of exposition we denote the state transition probability as

$$P(s^{'}|s,a) = P(s_{t+1} = s^{'}|(s_t = s, a_t = a))$$

where the pair (s,a) has fixed values. That is, we omit to represent the variables and focus on the values.
The state and reward transition probability is

$$P(s^{'}, r^{'}|s,a) = P\big(s_{t+1} = s^{'}, r_{t+1} = r^{'}|(s_t = s, a_t = a)\big)$$

From the marginal probability mass function (see Lemma (14.6.4)), we can get the probability to arrive at a state $s^{'}$ by summing over the possible rewards, getting

$$\begin{aligned} P(s^{'}|s,a) &= \sum_{r^{'} \in \mathcal{R}} P\big(s_{t+1} = s^{'}, r_{t+1} = r^{'}|(s_t = s, a_t = a)\big) \\ &= P(s_{t+1} = s^{'}|(s_t = s, a_t = a)) \end{aligned}$$

Following the sequential order in Statement (9.1.1) and using the Markov property (see Appendix (14.6.8)), the joint probability distribution of T random variables $s_1, s_2, ..., s_t$ (see Equation (14.6.14)) simplifies to

$$P(s_1, s_2, ..., s_T) = P(s_1) \times P(s_2|s_1) \times \cdots \times P(s_T|s_{T-1})$$

Adding the action, we get

$$P(s_1, a_1, ..., s_T, a_T) = P(s_1) \prod_{t=1}^{T} \pi(s_t, a_t) P(s_{t+1}|(s_t, a_t)) \tag{9.3.1}$$

We also have the relation

$$P(s_{t+1} = s^{'}, a_{t+1} = a^{'}|(s_t = s, a_t = a)) = P(s_{t+1} = s^{'}|(s_t = s, a_t = a)) P(a_{t+1} = a^{'}|s_{t+1} = s^{'})$$

Computing expectation by conditioning (see Equation (14.6.18)), we get the average reward, given a state and an action, as follows:

$$\begin{aligned} R(s,a) &= E[r_{t+1}|(s_t = s, a_t = a)] \\ &= \sum_{s^{'}, a^{'}} r(s^{'}, a^{'}) P(s^{'}, a^{'}|(s,a)) \end{aligned} \tag{9.3.2}$$

We can also get the conditional average reward as

$$\begin{aligned} R(s,a) &= E[r_{t+1}|(s_t = s, a_t = a)] \\ &= \sum_{r^{'}, s^{'}} r^{'} P(s^{'}, r^{'}|s,a) \end{aligned}$$

The state and reward sequence $(s_1, r_2), (s_2, r_3), ...$ is a Markov reward process $(\mathcal{S}, P_\pi, \mathcal{R}_\pi, \gamma)$ where the state transition probability, under the policy π, is

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} P_\pi(s^{'}|s) &= \sum_{a\in\mathcal{A}} P(a_t = a|s_t = s)P(s_{t+1} = s^{'}|(s_t = s, a_t = a)) \\ &= \sum_{a\in\mathcal{A}} P(a|s)P(s^{'}|(s,a)) \end{aligned}$$

Also, using the conditional expectation, under the policy π, we get

$$\begin{aligned} R_{\pi,s} = E_\pi[r_{t+1}|s_t = s] &= \sum_{a\in\mathcal{A}} E[r_{t+1}|(s_t = s, a_t = a)]P(a_t = a|s_t = s) \\ &= \sum_{a\in\mathcal{A}} P(a|s)R(s,a) = E_\pi[R(s,a)|s_t = s] \end{aligned}$$

Similarly, for a smooth function $f(\cdot)$, we get

$$\begin{aligned} F_{\pi,s} = E_\pi[f(s_{t+1})|s_t = s] &= \sum_{a\in\mathcal{A}} P(a_t = a|s_t = s)E[f(s_{t+1})|(s_t = s, a_t = a)] \\ &= \sum_{a\in\mathcal{A}} P(a_t = a|s_t = s) \sum_{s^{'}\in\mathcal{S}} f(s^{'})P(s_{t+1} = s^{'}|(s_t = s, a_t = a)) \\ &= \sum_{a\in\mathcal{A}} P(a|s) \sum_{s^{'}\in\mathcal{S}} f(s^{'})P(s^{'}|s,a) \\ &= \sum_{s^{'}\in\mathcal{S}} f(s^{'}) \sum_{a\in\mathcal{A}} P(a|s)P(s^{'}|s,a) \\ &= \sum_{s^{'}\in\mathcal{S}} f(s^{'})P_\pi(s^{'}|s) \end{aligned} \tag{9.3.3}$$

We can summarise the MDP as follows:

- Environment (E): $P(s^{'}|s,a)$ is the state transition matrix and $R(s,a) =< r|s,a >$ is the conditional average reward.
- Agent (A): $\pi(s,a) = P(a|s)$ is the policy.

9.3.1.2 The average reward

The aim of the agent is to maximise the average reward, that is, to select a policy maximising rewards. Thus, we need to find a way to define the average reward as we do not want the sum of all future rewards to be infinite (this is the assumption in Equation (14.7.26)). Several solutions exists such as

- Fixed finite horizon: We want to maximise the average

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T = \sum_{k=0}^{(T-t)-1} r_{t+1+k}$$

- Terminal state: The overall time may vary but it remains finite with probability 1. It is achieved in MDP with a terminal state denoted g, so that the following conditions apply

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} P(s^{'}|g,a) &= \delta_{s^{'},g} \\ P(s^{'},r^{'}|g,a) &= \delta_{r,0} \\ \lim_{t\to\infty} P(s_t = g) &= 1\ , \forall\pi \end{aligned}$$

In this setting, the goal is to maximise the expectation of

$$R_t = \sum_{k=0}^{\infty} r_{t+1+k}$$

- Discounted reward: At time t, the discounted reward (DR) is

$$\begin{aligned} R_t &= \sum_{k=0}^{\infty} \gamma^k r_{k+1+t} \\ &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \end{aligned} \tag{9.3.4}$$

where $0 < \gamma < 1$ is the discount rate. It is using the properties of geometric series [2] For small γ we ignore farther-away events, and vice versa for large γ.
We write the rate R_t as

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \\ &= r_{t+1} + \gamma\big(r_{t+2} + \gamma r_{t+3} + \cdots\big) \\ &= r_{t+1} + \gamma R_{t+1} \end{aligned} \tag{9.3.5}$$

We can multiply the DR with $alpha$, getting

$$\alpha R_t = \frac{1}{\text{norm}}\big(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots\big)$$

where $\alpha = \frac{1}{\text{norm}}$ and $\text{norm} = (1-\gamma)$

***Remark* 9.3.1** *Note, R_t is a random variable, the reward r_t at each stage is stochastic, depending on the behaviour of the environment and on the policy of the agent. Hence, we want to maximise the mean discounted reward R_t.*

It is a stochastic control problem (see Appendix (14.7) where the reward is the state of the system and the action is the control.
The value function $V(s)$ assigns a (long-term) value to each state. It is the average discounted reward, under the policy π, given that we start at state s. It is given by

$$V_\pi(s) = E_\pi[R_t|s_t = s] =< R_t|s_t = s >$$

It depends on the policy of the agent. Note, it corresponds to the performance function in Equation (14.7.25). The dynamics of MDP will cause the agent to wander in the state space, so that we need to be able to maximise the value function from all initial conditions.

[2] The geometric series states that for $|x| < 1$ we have $\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n$ which is an infinite series with finite sums. The telescoping series states that for $|x| < 1$ we define $g(K) = \frac{x^K}{1-x}$, then $g(K) \to 0$ as $K \to \infty$.

Electronic copy available at: https://ssrn.com/abstract=3307566

9.3.2 The Bellman equations

We assume that we use the discount reward in Equation (9.3.5) along with the Markov property. Using conditional expectation in Equation (9.3.2), we can rewrite the value function into two parts as

$$\begin{aligned} V_\pi(s) &= < R_t|s_t = s > \\ &= < r_{t+1} + \gamma R_{t+1}|s_t = s > \\ &= < r_{t+1} + \gamma V_\pi(s_{t+1})|s_t = s > \\ &= \sum_{a\in\mathcal{A}} \pi(s,a)\Big(R(s,a) + \gamma \sum_{s^{'}\in\mathcal{S}} P(s^{'}|s,a)V_\pi(s^{'})\Big) \end{aligned}$$

In the simple case where the action a is not a random variable (MRP), the above equation simplifies to

$$V(s) = R(s) + \gamma \sum_{s^{'}\in\mathcal{S}} P(s^{'}|s)V_\pi(s^{'})$$

where $R(s) =< r_{t+1}|s_t = s >$.
The Bellman equations are a set of linear equations having a unique solution for $0 < \gamma < 1$ (under mild conditions on the transition matrix). Thus, it can be solved by matrix inversion.
In the case where the action a is not a random variable, the Bellman equation can be expressed using matrices as

$$V_s = R_s + \gamma T_{ss'} V_s$$

where $V_s = \big(V(1), ..., V(n)\big)^\top$ is a column vector with one entry per state, $R_s = R(s)$ is a column vector of size n, and $T_{ss'} = P(s^{'}|s)$ is an $n \times n$ transition matrix. That is,

$$\begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} T_{11} & \cdots & T_{1n} \\ \vdots & & \vdots \\ T_{n1} & \cdots & T_{nn} \end{bmatrix} \begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix}$$

The value function becomes

$$V_s = \big(I - \gamma T_{ss'}\big)^{-1} R_s$$

In the case where the action is a random variable (the MDP), the Bellman equation can be expressed as

$$V_\pi = R_{\pi,s} + \gamma T_{\pi,ss'} V_\pi \tag{9.3.6}$$

where

$$T_{\pi,ss'} = \sum_{a\in\mathcal{A}} \pi(s,a)P(s^{'}|s,a) \text{ and } R_{\pi,s} = \sum_{a\in\mathcal{A}} \pi(s,a)R(s,a)$$

and the value function becomes

$$V_\pi = \big(I - \gamma T_{\pi,ss'}\big)^{-1} R_{\pi,s}$$

Since the computational cost of inverting matrices is large for a number of $O(n^3)$ where n is the number of states, one solution is to use an iterative approximation for V_π. Some iterative methods for large Markov reward process (MRP) and Markov decision process (MDP) are:

- Dynamic programming

Electronic copy available at: https://ssrn.com/abstract=3307566

- Monte Carlo evaluation
- Temporal difference (TD) learning

Iterative Policy Evaluation is an example of Dynamic Programming Algorithm which uses the Bellman equation as an update rule for successive approximations $V_0, V_1, V_2, ...$ of the true value function. We can start from arbitrary initial function $V_0(s)$ and compute the $k+1$ approximation as

$$V_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \Big(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_k(s') \Big)$$

Theorem 9.3.1 *The series of approximations converge asymptotically to* $V_\pi(s)$.

We will present these methods in Section (9.4).

9.3.3 A mathematical formalism

In the Markov decision process (MDP), the decision process is given in Algorithm (24).

Algorithm 24 Markov Decision Algorithm

Require: : training set S, number of training steps T
1: Initialise the state $s_0 \sim p(s_0)$
2: **for** $t = 0$ to $T-1$ **do**
3: Agent select action a_t
4: Environment samples reward $r_t \sim \mathcal{R}(\cdot|s_t, a_t)$
5: Environment samples next state $s_{t+1} \sim \mathbb{P}(\cdot|s_t, a_t)$
6: Agent receives reward r_t and next state s_{t+1}
7: **end for**

A policy π (see Definition (9.3.1)) is a function from $\mathcal{S}$ to $\mathcal{A}$, $\pi : \mathcal{S} \to \mathcal{A}$, that specifies what action to take in each state. The objective is to find a policy π^* that maximises the cumulative discounted reward

$$R_t = \sum_{t \geqslant 0} \gamma^t r_t$$

Note, this formulation of the cumulative discounted reward is slightly different from $R_t = \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k}$ where the first element is r_{t+1}.
We need to deal with the randomness from the initial state $s_0 \sim p(s_0)$, the action $a_t \sim \pi(\cdot|s_t)$, the next state $s_{t+1} \sim p(\cdot|s_t, a_t)$ and the transition probability. To do so, we let the objective $J(\pi)$ be the expected cumulative discounted reward

$$J(\pi) = E[\sum_{t \geqslant 0} \gamma^t r_t | \pi]$$

which we maximise as

$$\pi^* = \arg\max_{\pi} J(\pi)$$

From Algorithm (24) we see that following a policy, π, produces the trajectories (or paths)

$$(s_0, a_0, r_0), (s_1, a_1, r_1), ...$$

Electronic copy available at: https://ssrn.com/abstract=3307566

By analogy with stochastic control, these trajectories correspond to the state of the system, and the action a corresponds to the stochastic control u.
Assuming fixed maturity T and $\gamma = 1$, from Equation (9.3.1), we can write the joint mass function as

$$\begin{aligned} p(s_0, a_0, s_1, a_1, ..., s_T, a_T) = p(\tau) &= \prod_{t \geqslant 0} P(s_{t+1}|(s_t, a_t))\pi(s_t, a_t) \\ &= p(s_0)\pi(s_0, a_0) \prod_{t \geqslant 1} P(s_{t+1}|(s_t, a_t))\pi(s_t, a_t) \end{aligned}$$

where τ is a trajectory. We can rewrite the optimisation problem with respect to the density $p(\tau)$ as

$$\pi^* = \arg\max_{\pi} E_{\tau \sim p(\tau)}[\sum_{t \geqslant 0} r(s_t, a_t)] \, , r_t = r(s_t, a_t)$$

where $E_{\tau \sim p(\tau)}[\cdot]$ indicates that actions are sampled from π to generate the trajectory $\tau = (s_0, a_0, s_1, a_1, ...)$. The exponential operator being a linear operator, the optimisation problem can be written as

$$\pi^* = \arg\max_{\pi} \sum_{t \geqslant 0} E_{(s_t, a_t) \sim p(s_t, a_t)}[r(s_t, a_t)]$$

This expectation is expressed with respect to the state-action pair (s_t, a_t) and the joint distribution $p(s_t, a_t)$. Following the sequential order in Statement (9.1.1), we have $s_t \sim p(s_t)$ and $a_t \sim \pi(s_t, a_t)$, so that the expectation can be decomposed as

$$E_{\tau \sim p(\tau)}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)] = \sum_{t=0}^{\infty} E_{s_t \sim p(s_t)}[E_{a_t \sim \pi(a_t, s_t)}[\gamma^t r(s_t, a_t)]]$$

which is a sum of conditional expectations.
If we write the future reward from time $t = 1$ as

$$Q(s_1, a_1) = r(s_1, a_1) + E_{s_2 \sim p(s_2)}[E_{a_2 \sim \pi(a_2, s_2)}[r(s_2, a_2) + \cdots | s_2] | (s_1, a_1)]$$

then we get

$$E_{s_1 \sim p(s_1)}[E_{a_1 \sim \pi(a_1, s_1)}[Q(s_1, a_1)] | s_1]$$

If we know $Q(s_1, a_1)$ we can modify the policy $\pi(a_1, s_1)$ to improve it further. For example, a well known policy is

$$\pi(a_1, s_1) = 1 \text{ if } a_1 = \arg\max_{a_1} Q(s_1, a_1)$$

Alternatively, we can compute the gradient to increase the probability of getting good actions. If we define the average $V_\pi(s) = E_\pi[Q_\pi(s, a)]$, and if $Q_\pi(s, a) > V_\pi(s)$, then the action is better than average. Consequently, we can modify π to improve the action. It leads to the approach of the value function and Q-value function which we are now going to present.

9.3.3.1 The value function

Definition 9.3.2 *The value function of an MDP at state s is the expected cumulative reward from following the policy from state s:*

$$V_\pi(s) = E[\sum_{t \geqslant 0} \gamma^t r_t | s_0 = s, \pi]$$

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, this function corresponds to the performance function in Equation (14.7.25) with $K(\cdot) = 0$, the continuous integral is replaced by a discrete one, and the utility function is linear with respect to the reward r.

***Remark* 9.3.2** *The value function is conditioned to the state s (fixed). The action is a random variable.*

Thus, using the conditional expectation (see Equation (9.3.3)), we can rewrite the value function as

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) E_\pi[\sum_{t \geqslant 0} \gamma^t r_t | (s_0 = s, a_0 = a)] \tag{9.3.7}$$

The value function can be decomposed into immediate reward plus discounted value of successor states

$$V_\pi(s) = E_\pi[r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s] \tag{9.3.8}$$

which we can write as

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \Big(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_\pi(s') \Big)$$

The learning goal is to obtain the best possible value for each initial state, that is, to obtain the optimal value function.

Definition 9.3.3 *The optimal value function V^* is the maximum expected cumulative reward achievable from a given state over all policies:*

$$V^*(s) = \max_\pi V_\pi(s)$$

An MDP is solved once we know the optimal value function V^*. Thus, we need to find π^* such that

$$V_{\pi *}(s) = V^*(s) \, , \forall s$$

Such a policy is an optimal policy since it satisfies

$$V_{\pi *}(s) \geqslant V_\pi(s) \, , \forall \pi, s$$

As discussed in Appendix (14.7), under some conditions, such a policy exists. The optimal value function $V^*(s)$ satisfies the the Bellman optimality equation (BOE) (value function in Appendix (14.8.30))

$$V^*(s) = \max_a \big\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \big\} \, , \forall s \tag{9.3.9}$$

If $\mathcal{S}$ and $\mathcal{A}$ are finite set and $0 < \gamma < 1$, then the above equation has a unique solution.

9.3.3.1.1 The optimal policy The optimal policy, satisfying $V_{\pi *} = V^*$ is the deterministic policy given by

$$\pi(s,a) = \delta_{a,a(s)} \tag{9.3.10}$$

where $\delta_{a,b} = 1$ if $a = b$, and zero otherwise, and

$$a(s) = \arg\max_a \big\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \big\} \, , \forall s$$

Proof: To prove the existence of a solution to the BOE we start with a policy $\pi \Rightarrow V_\pi$. We then define a new policy π' as follows:

- π' is deterministic: $\pi'(s,a) = \delta_{a,a(s)}$

Electronic copy available at: https://ssrn.com/abstract=3307566

- $a(s) = \arg\max_a \{R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_\pi(s')\}$

We use that new policy to obtain a new value function $V_{\pi'}(s)$. Then, we need to prove that $V_{\pi'}(s) \geqslant V_\pi(s)$, for all s. We have

$$V_\pi(s) \leqslant E_{\pi'}[r_{t+1} + \gamma V_\pi(s_{t+1})|s_t = s], \forall s$$

which implies

$$\begin{aligned} V_\pi(s) &\leqslant E_{\pi'}[r_{t+1} + \gamma V_\pi(s_{t+1})|s_t = s] \\ &= E_{\pi'}[r_{t+1} + \gamma E_\pi[r_{t+2} + \gamma V_\pi(s_{t+2})]|s_t = s] \\ &\leqslant E_{\pi'}[r_{t+1} + \gamma E_{\pi'}[r_{t+2} + \gamma V_\pi(s_{t+2})]|s_t = s] \\ &= E_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 V_\pi(s_{t+2})|s_t = s] \\ &\leqslant E_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots |s_t = s] \\ &= V_{\pi'}(s) \end{aligned}$$

so that the new policy improves the previous one. At each step, we move in the most greedy manner to improve the policy. This iterative process is called Greedy Policy Improvement. Allowing for a finite number of states (s,a), then $V_\pi(s)$ is bounded from above, allowing for convergence to a finite value. Thus, $\pi \to \pi^\infty$ then $V_\pi \to V_\infty$ which satisfy the Bellman optimality equation. Further, $V_\infty = V_{\pi^\infty}$, where

$$\pi^\infty(s,a) = \delta_{a,a(s)}$$

and

$$a(s) = \arg\max_a \{R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_\infty(s')\}$$

Note, $V_\infty(s)$ can be a local maximum which also satisfies the BOE. One can prove that there is a single value function satisfying that equation.

9.3.3.2 The Q-value function

Similarly to the value function which assigns values to states, we can also write a value function of states and actions. It is called the Q-value function.

Definition 9.3.4 *The Q-value function at state s and action a is the expected cumulative reward from taking action a in state s and then following the policy π:*

$$Q_\pi(s,a) = E[\sum_{t \geqslant 0} \gamma^t r_t | (s_0 = s, a_0 = a), \pi]$$

***Remark* 9.3.3** *The Q-value function summarises the performance of each action from a given state, assuming it follows π afterwards. It is conditioned to the fixed pair (s,a). There is no need to sum over all possible actions.*

***Remark* 9.3.4** *Williams [1992] suggested that $Q_\pi(s,a)$ could be approximated with the actual returns $R_t = \sum_{t \geqslant 0} \gamma^t r_t$. This is a property of ergodic processes (see Appendix (14.6.9)).*

Electronic copy available at: https://ssrn.com/abstract=3307566

The Q-value function can be decomposed into immediate reward plus discounted value of successor states

$$Q_\pi(s,a) = E_\pi[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1})|(s_t = s, a_t = a)] \tag{9.3.11}$$

which we can write as

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s^{'} \in \mathcal{S}, a^{'} \in \mathcal{A}} P(s^{'}|(s,a))\pi(s^{'}, a^{'})Q_\pi(s^{'}, a^{'})$$

or more simply as

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s^{'} \in \mathcal{S}, a^{'} \in \mathcal{A}} P(s^{'}, a^{'}|(s,a))Q_\pi(s^{'}, a^{'})$$

Further, given Equation (9.3.7), we can write the value function $V(s)$ in terms of the Q-value function as

$$\begin{aligned} V_\pi(s) &= \sum_{a \in \mathcal{A}} \pi(s,a) Q_\pi(s,a) \\ &= E_{a_t \sim \pi(s_t, a_t)}[Q_\pi(s_t, a_t)] \end{aligned} \tag{9.3.12}$$

so that the Q-value function becomes

$$\begin{aligned} Q_\pi(s,a) &= R(s,a) + \gamma \sum_{s^{'} \in \mathcal{S}} P(s^{'}|s,a) V_\pi(s^{'}) \\ &= R(s,a) + \gamma E_{s_{t+1} \sim p(s_{t+1}|(s_t, a_t))}[V_\pi(s_{t+1})] \end{aligned} \tag{9.3.13}$$

Definition 9.3.5 *The optimal Q-value function Q^* is the maximum expected cumulative reward achievable from a given $(state, action)$ pair over all policies:*

$$Q^*(s,a) = \max_\pi Q_\pi(s,a)$$

Thus, we need to find π^* such that

$$Q_{\pi^*}(s,a) = Q^*(s,a)$$

Since it is easier to work with Q than V, then given Q^* we can infer V^* as follows:

$$V^*(s) = \max_a Q^*(s,a) \tag{9.3.14}$$

which is

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s^{'} \in \mathcal{S}} P(s^{'}|s,a) V^*(s^{'})$$

Putting terms together, we can write the optimal Q-value function as

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s^{'} \in \mathcal{S}} P(s^{'}|s,a) \max_{a^{'}} Q^*(s^{'}, a^{'})$$

From the above equation and the principle of dynamic programming (see Appendix (14.8.3.1)), Q^* satisfies the Bellman optimal equation (BOE)

Electronic copy available at: https://ssrn.com/abstract=3307566

$$Q^*(s,a) = E_{s' \sim p(s'|s,\pi(s))}[r + \gamma \max_{a'} Q^*(s', a')|s, a] \tag{9.3.15}$$

It states that if the optimal state-action values for the next time step $Q^*(s', a')$ are known, then the optimal strategy is to take the action that maximises the expected value of $r + \gamma Q^*(s', a')$.

***Remark* 9.3.5** *The optimal policy π^* corresponds to taking the best action in any state as specified by Q^*. Thus, we want a greedy policy π^* that always seeks an action maximising the Q-value function in the current state:*

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a)$$

9.4 Learning the optimal policy

The goal is to learn the optimal policy π^*, or, alternatively, the optimal value function $V^*(s)$. We can always derive π^* from $V^*(s)$ and vice versa, since the optimal policy satisfies $s \to a^*(s)$ where the optimal action $a^*(s)$ is defined above (see Equation (9.3.10)). That is, learning optimal actions for all states simultaneously means learning a policy, which is our objective.
In general there is no closed form solution to the Bellman equation and one must rely on some iterative solutions such as:

- Value Iteration
- Policy Iteration
- Q-learning
- Sarsa
- Dynamic programming

Standard function approximations approximate the value function and determine a policy from it. However, it is oriented towards finding deterministic policies, it suffers from discontinuous changes in the estimated value making it difficult to converge. An alternative solution is to let the policy be explicitly represented by its own function approximator, and update it according to the gradient of expected reward with respect to the policy parameters.
We classify the algorithms for policy optimisation into three broad categories:

1. Policy iteration methods: they alternate between estimating the value function under the current policy and improving the policy (see Bertsekas [2005]).
2. Policy gradient methods: they use an estimator of the gradient of the expected cost obtained from sample trajectories (Peters et al. [2008]).
3. Derivative-free stochastic optimisation methods: examples are the cross-entropy method (CEM) and covariance matrix adaptation (CMA), which treat the cost as a black box function to be optimised in terms of the policy parameters (see Fu et al. [2005]). These methods are very good for dynamic programming and continuous control problems.

These methods implement a single-step update of a current value of the Bellman equation. Since the state space and action space are both discrete, we get a value for each combination of a state and action. As such the MDP is represented in a tabulated form as a two-dimensional matrix (tensor). For a time-dependent problem, a time argument is added to the function so that the table is represented by a three-dimensional tensor for a discrete state-action space. We are now going to present some of these methods.

Electronic copy available at: https://ssrn.com/abstract=3307566

9.4.1 The policy iteration

9.4.1.1 Introduction

In dynamic programming (DP) we assume that we know the transition matrix of the environment $P(s^{'}|s,a)$, for discrete state-action pair (s,a), and the expected next step reward. Given Remark (9.3.5), we want to find a greedy policy π^* that always seeks an action maximising the Q-value function in the current state.
Given Equation (9.3.14), one solution is to let the Advantage Function A_π define how much an action a is better than the expected action according to the policy π (see details in Section (9.4.4)). It is given by

$$\begin{aligned} A_\pi(s,a) &= Q_\pi(s,a) - V_\pi(s) \\ &= r(s,a) + \gamma E[V_\pi(s^{'})] - V_\pi(s) \end{aligned}$$

The main idea behind policy iteration follows the Greedy Policy Improvement. It can be described with the following algorithm:

1. Estimate $A_\pi(s,a)$.
2. Set $\pi \leftarrow \pi^{'}$ (policy update).
3. Repeat step [1].

where

$$\pi^{'}(a,s) = \begin{cases} 1 \text{ if } a = \arg\max_a A_\pi(s,a) \\ 0 \text{ otherwise} \end{cases}$$

Thus, we need to estimate the value function $V_\pi(s)$.
We can also replace the optimisation of the Advantage Function with that of the Q-value function $Q_\pi(s,a)$, getting $\arg\max_a Q_\pi(s,a)$. In that case, we get the algorithm

1. Set $Q(s,a) \leftarrow r(s,a) + \gamma E[V_\pi(s^{'})]$
2. Set $V_\pi(s) \leftarrow \max_a Q(s,a)$
3. Repeat step [1].

Some examples of policy are

- Final policy
$$\pi(a,s) = \begin{cases} 1 \text{ if } a = \arg\max_a Q_\phi(s,a) \\ 0 \text{ otherwise} \end{cases}$$
- Epsilon-greedy
$$\pi(a,s) = \begin{cases} 1-\epsilon \text{ if } a = \arg\max_a Q_\phi(s,a) \\ \frac{\epsilon}{(|\mathcal{A}|-1)} \text{ otherwise} \end{cases}$$
- Boltzmann exploration
$$\pi(a,s) \propto e^{Q_\phi(s,a)}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

Given the Bellman optimality equation (BOE) in Equation (9.3.9) and, using the matrix notation in Section (9.3.2), we can define an operator $\mathcal{B}$ as follows:

$$\mathcal{B}V = \max_a \big(r_a + \gamma \mathcal{T}_a V\big)$$

where r_a is a vector of rewards at all states for action a and $\mathcal{T}_a = P(s^{'} = i|s = j, a)$ is the transition matrix for action a. Since

$$V^*(s) = r(s, a) + \gamma E[V^*(s^{'})]$$

so $V^* = \mathcal{B}V^*$ and the optimum V^* is a fixed point of $\mathcal{B}$. Since $\mathcal{B}$ is a contraction, one can prove that the value iteration reaches V^*.

9.4.1.2 Value Iteration

A simple example of Value Iteration algorithm uses the Bellman optimality equation (BOE) Bellman optimality equation (BOE) (value function in Appendix (14.8.3.1)). In the case of the value function with BOE in Equation (9.3.9), the evaluation is given by

$$V_\pi(s) \leftarrow r(s, \pi(s)) + \gamma E_{s' \sim p(s'|s,\pi(s))}[V_\pi(s^{'})]$$

and the update is obtained by maximising the policy. We start with some initial function V_0^*, and compute at each step the estimate

$$V_{i+1}^*(s) = \max_a \big[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s^{'}|s, a) V_i^*(s^{'})\big]$$

which converges asymptotically to V^*.
Similarly for the Q-value function with BOE in Equation (9.3.15), starting with some initial function Q_0^*, the iterative update is given by

$$\begin{aligned} Q_{i+1}^*(s, a) &= E\big[r + \gamma \max_{a'} Q_i^*(s^{'}, a^{'})|s, a\big] \\ &= R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s^{'}|s, a) \max_{a'} Q_{i+1}^*(s^{'}, a^{'}) \end{aligned}$$

and Q_i^* will converge to Q^* as $i \to \infty$.
However, this algorithm is not scalable as one must compute $Q^*(s, a)$ for every state-action pair, which is computationally infeasible to compute for the entire state space. Further, we are required to know both the transition matrix of the environment and the expected next step reward.

9.4.1.3 TD learning

One solution to the above problem is to consider on-line learning rules called Temporal Difference (TD) learning. At each time step t, we assume no information is given about the environment and the sequence $s_t, a_t \to r_{t+1}, s_{t+1}$ is all the information known. That is, it does not make any assumption on the true data that generated the observation $\big(s_t, a_t, r_t, s_{t+1}\big)$. This is called model-free learning.
A naive estimate would be to replace the full value iteration algorithm

$$V_{i+1}(s) = \sum_a \pi(s, a)\big[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s^{'}|s, a) V_i(s^{'})\big]$$

Electronic copy available at: https://ssrn.com/abstract=3307566

with the approximation

$$V_{t+1}(s_t) = r_{t+1} + \gamma V_t(s_{t+1})$$

Given $V_\pi(s)$ in Equation (9.3.8), as $t \to \infty$, one can show that the estimate converges. However, even though this algorithm will improve the estimate of V on average, we can not control its variance. One way around is to introduce a learning rate parameter $\alpha_t > 0$ to produce small steps. We let the TD error be

$$\sigma_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

which corresponds to the Advantage Function. Taking the expectation of the error signal, we get

$$E[\sigma_t] = V_\pi(s) - V_t(s_t)$$

which is an instantaneous estimate of the deviation of the current estimate of V from its true value.
The update equation of the estimate is given by

$$V_{t+1}(s_t) = V_t(s_t) + \alpha_t \sigma_t$$

which we can rewrite as

$$V_{t+1}(s_t) = (1 - \alpha_t)V_t(s_t) + \alpha_t\big(r_{t+1} + \gamma V_t(s_{t+1})\big)$$

Hence, it corresponds to the stochastic approximation (see Appendix 14.5.5). Similarly for the Q-value function, a naive update is

$$Q_{t+1}(s_t, a_t) = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1})$$

but it will generate a lot of variance. We consider

$$\sigma_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$$

and the update rule is

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \sigma_t$$

9.4.1.4 Fitted value iteration

As discussed above, we have assumed that the transition matrix of the environment and the expected next step reward were known. If we do not know the transition dynamics, we can fit $V_\pi(s)$ to a model by using samples. We use a function approximator to estimate the value function $V(s)$, such as neural networks (see Remark (8.1.1)). It uses a function approximator to estimate the value function

$$V(s; \phi) \approx V^*(s)$$

where ϕ is the function parameters (or weights). We consider a neural network representation of value functions and back propagate the error. It is given by

$$\mathcal{L}(\phi) = \frac{1}{2}\|V_\phi(s) - \max_a Q_\pi(s, a)\|^2$$

The fitted value iteration algorithm is as follows:

1. Set $y_i \leftarrow \max_{a_i}\big(r(s_i, a_i) + \gamma E[V_\phi(s_i^{'})]\big)$
2. Set $\phi \leftarrow \arg\min_\phi \frac{1}{2}\sum_i \|V_\phi(s_i) - y_i\|^2$

Electronic copy available at: https://ssrn.com/abstract=3307566

3. Repeat step [1].

In the case of the Q-value function, we get the Q-Iteration algorithm:

1. Set $y_i \leftarrow r(s_i, a_i) + \gamma E[V_\phi(s_i^{'})]$.
2. Set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$.
3. Repeat step [1].

where we use the approximation $E[V_\phi(s_i^{'})] \approx \max_{a'} Q_\phi(s_i^{'}, a^{'})$. However, convergence is not guaranteed.
An alternative solution is the online Q-Iteration algorithm (see Ernst et al. [2005]):

1. Take some action a_i and observe $(s_i, a_i, s_i^{'}, r_i)$.
2. Set $y_i \leftarrow r(s_i, a_i) + \gamma E[V_\phi(s_i^{'})]$.
3. Set $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)\big(Q_\phi(s_i, a_i) - y_i\big)$ where $\widehat{y}_i = Q_\phi(s_i, a_i)$.
4. Repeat step [1].

Note, the observations can be correlated and step [3] is not equivalent to the gradient descent since there is no gradient through the target value y_i.
This is the Q-learning and its variants, which we will detail below.

9.4.2 The Q-learning

9.4.2.1 Q-learning

The Q-learning algorithm is a model-free on-line learning algorithm based on learning the optimal policy (see Watkins [1989] and Watkins et al. [1992]). One has some estimate of the Q-function $\{Q_t(s, a)\}$ which changes as new information is revealed. Similarly to the TD learning algorithm, it is a model-free and off-line policy. Given the data point $\big(s_t, a_t, r_t, s_{t+1}\big)$, the error signal is given by

$$\sigma_t = r_{t+1} + \gamma \max_{a^{'}} Q_t(s_{t+1}, a^{'}) - Q_t(s_t, a_t)$$

Given the optimal $Q^*(s, a)$ in Equation (9.3.15), taking the expectation of the error signal we get

$$E[\sigma_t] = Q^*(s, a) - Q_t(s_t, a_t)$$

so that if the estimate is optimal, the expected value of the error signal is null.
The iterative update is given by

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \sigma_t$$

which we can rewrite as

$$Q_{t+1}(s_t, a_t) = (1 - \alpha_t) Q_t(s_t, a_t) + \alpha_t \big(r_{t+1} + \gamma \max_{a^{'}} Q_t(s_{t+1}, a^{'})\big)$$

Hence, it corresponds to the stochastic approximation (see Appendix 14.5.5). As discussed above, if the estimate is the optimal Q, then Q_t should be equal to the average of the future reward:

$$Q_t(s_t, a_t) = r_{t+1} + \gamma \max_{a^{'}} Q_t(s_{t+1}, a^{'})$$

Electronic copy available at: https://ssrn.com/abstract=3307566

so that the update becomes

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t)$$

as expected. That is, Q^* is a fixed point.
We get the following convergence (see Gosavi [2014]):

Theorem 9.4.1 *For a small enough α, or for $\alpha \to 0$ as $t \to \infty$:*

$$\lim_{t \to \infty} Q_t(s_t, a_t) = Q^*(s, a)$$

This convergence is independent of the policy provided that we have an exhaustive sampling of state-action pairs (s, a). Once we know the optimal Q^* we also know the optimal value function V^* and the optimal policy π^*. This is an implicit way of learning the optimal policy.

9.4.2.2 Deep Q-learning

A better solution is to use a function approximator to estimate the Q-value function $Q(s, a)$, such as neural networks (see Remark (8.1.1)). It uses a function approximator to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

where θ is the function parameters (or weights). In order to deal with a large number of state-action pairs (s, a) one solution is to consider a neural network representation of value functions and back propagate the error. When the approximator is a deep NN it is called deep Q-learning. We want to find a Q-function that satisfies the BOE in Equation (9.3.15). Thus, we get the forward and backward pass:

1. The forward pass: Define the loss function

$$L_i(\theta_i) = E_{s,a \sim \rho(\cdot)}[\big(y_i - Q(s, a; \theta_i)\big)^2]$$

where

$$y_i = E_{s' \sim p(s')}[r + \gamma \max_{a'} Q^*(s', a'; \theta_{i-1})|s, a]$$

We iteratively try to make the Q-value close to the target value y_i (calibration) it should have if the Q-function corresponds to the optimal Q^* and optimal policy π^*.

2. The backward pass: The gradient update (with respect to θ)

$$\nabla_{\theta_i} L_i(\theta_i) = E_{s,a \sim \rho(\cdot), s' \sim p(s')}[\Big(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\Big) \nabla_{\theta_i} Q(s, a; \theta_i)|s, a]$$

For instance, in the case of Q-learning, we can choose a DNN with inputs being vector representation of the state s and outputs are multiple nodes representing $Q(s, a|\theta)$. After presentation of a state s, applying a greedy policy on the output Q, we get the reward and the new state and compute the error signal

$$\sigma_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a'|\theta) - Q_t(s_t, a_t|\theta)$$

If we let σ^2 be the objective function, taking the expectation we recover the above loss function. We can then get the gradient with respect to θ as

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\nabla_\theta \sigma^2 \sim -\sigma_t \nabla_\theta Q_t(s_t, a_t|\theta)$$

and the update of the parameter is

$$\theta^{'} = \theta - \beta_t \sigma_t \nabla_\theta Q_t(s_t, a_t|\theta)$$

9.4.2.3 Experience replay

When training the Q-network, it is problematic to learn from batches of consecutive samples because:

- samples are correlated, leading to inefficient learning.
- the current Q-network parameters determine the next training samples, which might lead to poor feedback loops.

One solution is to use experience replay:

- continuously update a replay memory table of transitions (s_t, a_t, r_t, s_{t+1}) as episodes are played.
- each transition can contribute to multiple weight updates to get greater data efficiency.
- train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples.

The classic deep Q-learning algorithm is as follows:

1. Take some action a_i and observe $(s_i, a_i, s_i^{'}, r_i)$, add it to $\mathcal{B}$.
2. Sample mini-batch $\{s_j, a_j, s_j^{'}, r_j\}$ from $\mathcal{B}$ uniformly.
3. Compute $y_j = r_j + \gamma \max_{a_j^{'}} Q_{\theta'}(s_j^{'}, a_j^{'})$ using target network $Q_{\theta'}$.
4. Update $\theta \leftarrow \theta - \beta \sum_j \nabla Q_\theta(s_j, a_j)\big(Q_\theta(s_j, a_j) - y_j\big)$
5. Update $\theta^{'}$, copy θ every N steps.

See details of the deep Q-learning with experience replay in Algorithm (25).

Algorithm 25 Deep Q-learning with Experience Replay Algorithm

Require: : training set S, number of training steps T

Initialise replay memory $\mathcal{D}$ to capacity N
Initialise action-value function Q with random weights
for $episode = 1$ to M **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi_1(s_1)$
  for $t = 1$ to $T - 1$ **do**
    With probability ϵ select a random action a_t, otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_s$, a_t, x_{t+1} and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set

$$y_j = \begin{cases} r_j \text{ for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a^{'}; \theta) \text{ for non-terminal } \phi_{j+1} \end{cases}$$

    Perform a gradient descent step on $\big(y_j - Q(\phi_j, a_j; \theta)\big)^2$
  end for
end for

Electronic copy available at: https://ssrn.com/abstract=3307566

9.4.2.4 The continuous state-action method

In the discrete-state models considered so far, both the value function the policy function can be stored in a tabulated form. This is not the case for continuous-state models where one has to rely on function approximations to represent compactly the value function, or the policy function, or both. One solution is to use linear architectures expressing the function as a linear combination

$$f(x) = \sum_k \theta_k \phi_k(x)$$

for some fixed basis functions $\{\phi_k(x); k = 1, ..., K\}$. It replaces the problem of general function approximation in a non-parametric setting by a linear parametrised function approximation, making it easier to solve as we only need to fit the parameters $\{\theta_k\}$. However, it is not an easy task to find a good set of basis functions as well as the optimal number of such functions. This method has been used in finance in the pricing of American options with Monte Carlo. This is the Least Square Monte Carlo (LSMC) method (see Longstaff et al. [2001]). Using similar techniques, the Q-learning was extended to continuous state-action cases in Fitted Q-Iteration (FQI) method (see Ernst et al. [2005], Murphy [2005]). Note that the discrete-state case can be considered as a special case of the continuous-state formulation.

9.4.3 The policy gradients

One problem with Q-learning is that even though the Q-function may be very complex, the policy can be much simpler. So, we could directly learn the latter from a collection of policies. Thus, we let the policy be explicitly represented by its own function approximator, and update it according to the gradient of expected reward with respect to the policy parameters. The main idea being to directly learn the policy with a neural network, since they are function approximator (see Remark (8.1.1)), and use an independent estimate of the value of a policy for an update rule. The REINFORCE algorithm and actor-critic methods are examples of this approach.
To do so, we define a class of parametrised policies as

$$\prod = \{\pi_\theta, \theta \in \mathbb{R}^m\}$$

where θ is the policy parameters. An example of a parametric policy, the Gibbs distribution in a linear combination of features is given by

$$\pi_\theta(s, a) = \frac{e^{\beta\theta^\top f(s,a)}}{\sum_{a' \in \mathcal{A}} e^{\beta\theta^\top f(s,a')}}, \forall s \in \mathcal{S}, a \in \mathcal{A}$$

where f is some d-dimensional feature basis for representation of the pairs (s, a). In order to learn θ we need an objective function for π. For each policy, we define the value function as

$$J(\theta) = E[\sum_{t \geqslant 0} \gamma^t r_t | \pi_\theta] \tag{9.4.16}$$

We want to find the optimal policy

$$\theta^* = \arg\min_\theta J(\theta)$$

One solution is to use the gradient ascent on policy parameters θ. It leads to the REINFORCE algorithm, which we have described in Section (9.2) in the case of stationary reward, where $< r >= R(W)$. In presence of stochastic reward, one can use some Monte Carlo (MC) (see Appendix (15.2)) policy gradient methods (see Peters et al. [2006]) where the objective function can be approximated as

$$J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t \geqslant 0} \gamma^t r_{t,i}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

They are on-policy methods directly maximising the cumulative future returns with respect to the policy. However, unless control variate, or other variance reduction methods (see Appendix (15.2.2)), is applied, MC simulations suffer from high variance. We distinguish between methods using full MC returns, such as REINFORCE (see Williams [1992]), and methods with function approximation such as actor-critic methods (see Sutton et al. [1999]) which optimise the policy against a critic. In the latter, the gradient is estimated with the help of an approximate Q-value function or Advantage Function (the critic).

9.4.3.1 The REINFORCE algorithm

We use the notation in Section (9.3.3), where τ is a trajectory and $p(\tau)$ is the joint mass function. We denote $R(\tau) = \sum_{t\geqslant 0} r(s_t, a_t)$ the total reward for the trajectory $\tau = (s_0, a_0, r_0), (s_1, a_1, r_1), ...$ In the setting of the value function, we can write the expected reward as

$$\begin{aligned} J(\theta) &= E_{\tau\sim p(\tau;\theta)}[R(\tau)] \\ &= \int_\tau R(\tau)p(\tau;\theta)d\tau \end{aligned}$$

where $p(\cdot;\theta)$ is the density function of that trajectory. We can differentiate the value function with respect to the policy parameters θ, getting

$$\nabla_\theta J(\theta) = \int_\tau R(\tau)\nabla_\theta p(\tau;\theta)d\tau$$

which is intractable when the density p depends on θ. However, we can proceed as follows:

$$\begin{aligned} \nabla_\theta p(\tau;\theta) &= p(\tau;\theta)\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} \\ &= p(\tau;\theta)\nabla_\theta \log p(\tau;\theta) \end{aligned} \tag{9.4.17}$$

It acts like a change of probability measure. Plugging it back into the gradient, we get

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_\tau \Big(R(\tau)\nabla_\theta \log p(\tau;\theta)\Big)p(\tau;\theta)d\tau \\ &= E_{\tau\sim p(\tau;\theta)}[R(\tau)\nabla_\theta \log p(\tau;\theta)] \end{aligned}$$

There are several ways for estimating this expectation among which is the Monte Carlo sampling (see Appendix (15.2)).
We can write the density function for the trajectory τ as

$$p(\tau;\theta) = \prod_{t\geqslant 0} p(s_{t+1}|s_t, a_t)\pi_\theta(s_t, a_t)$$

Thus, taking the logarithm, we get

$$\log p(\tau;\theta) = \sum_{t\geqslant 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(s_t, a_t)$$

Differentiating with respect to θ, we get

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geqslant 0} \nabla_\theta \log \pi_\theta(s_t, a_t)$$

Electronic copy available at: https://ssrn.com/abstract=3307566

which does not depend on the transition probabilities $p(s_{t+1}|s_t, a_t)$. Replacing in the gradient, we get

$$\nabla_\theta J(\theta) = E_{\tau\sim p(\tau;\theta)}[R(\tau)\sum_{t\geqslant 0}\nabla_\theta \log \pi_\theta(s_t, a_t)]$$

As a result, when sampling a trajectory τ (sample $\{\tau_i\}$ from $\pi_\theta(s_t, a_t)$), we can estimate the performance $J(\theta)$ with the gradient estimator

$$\nabla_\theta J(\theta) \approx R(\tau)\sum_{t\geqslant 0}\nabla_\theta \log \pi_\theta(s_t, a_t)$$

That is, when $R(\tau)$ is high, it push up the probabilities of the action seen, while when it is low the probabilities are pushed down.

***Remark* 9.4.1** *In the case where we consider all the sampling trajectories $\{\tau_i\}_{i=1}^N$, the gradient estimator becomes*

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^N\Big(\sum_{t\geqslant 0}\nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t})\Big)R(\tau_i)$$

where $R(\tau_i) = \sum_{t\geqslant 0} r(s_{t,i}, a_{t,i})$. The maximum likelihood is

$$\nabla_\theta J_{ML}(\theta) \approx \frac{1}{N}\sum_{i=1}^N\sum_{t\geqslant 0}\nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t})$$

To simplify notation we write

$$\nabla_\theta \log \pi_\theta(\tau_i) = \sum_{t\geqslant 0}\nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t})$$

so that the gradient estimator and the ML gradient simplify to

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^N\nabla_\theta \log \pi_\theta(\tau_i)R(\tau_i) \text{ and } \nabla_\theta J_{ML}(\theta) \approx \frac{1}{N}\sum_{i=1}^N\nabla_\theta \log \pi_\theta(\tau_i)$$

The extra term makes the good trajectories more likely and the bad ones less likely. However, it is problematic in the case of good trajectories when $R(\tau)$ is negligible.
In this simplified notation the optimisation problem is

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[R(\tau)]$$

with gradient

$$\nabla_\theta J(\theta) = E_{\tau\sim p(\tau;\theta)}[\nabla_\theta \log \pi_\theta(\tau)R(\tau)]$$

***Remark* 9.4.2** *Note that this approach suffers from high variance since the credit assignment is hard.*

We must therefore find a way to reduce that variance.

Electronic copy available at: https://ssrn.com/abstract=3307566

9.4.3.1.1 Example As an example, we can consider Gaussian policies $\pi_\theta(s_t, a_t) = N(\mu_\theta(s_t), \Sigma_\theta)$ where $\mu_\theta(s_t) = E_{\pi_\theta}[a_t] = f_{network}(s_t)$ is the mean of the policy and $N(\cdot, \cdot)$ is the normal cumulative distribution function. We get

$$\log \pi_\theta(s_t, a_t) = -\frac{1}{2}\|f_{network}(s_t) - a_t\|^2 + C$$

where C is a constant. Its derivative is

$$\nabla_\theta \log \pi_\theta(s_t, a_t) = -\frac{1}{2}\Sigma_\theta^{-1}\big(f_{network}(s_t) - a_t\big)\frac{df}{d\theta}$$

9.4.3.2 Variance reduction

The gradient is estimated by using Monte Carlo (MC) samples, which can have high variance. We discuss some solutions to improve the policy gradient by reducing the variance (see Appendix (15.2.2)). For simplicity of exposition we consider sampling a trajectory τ rather than looking at all trajectories at once.

9.4.3.2.1 Some solutions Some solutions of variance reduction are:

1. The policy at time $t^{'}$ does not affect reward at time t when $t < t^{'}$. Thus, rather than using the total reward from inception $R(\tau) = \sum_{t \geqslant 0} r(s_t, a_t)$ we consider the reward to go $\sum_{t^{'} \geqslant t} r_{t^{'}}$ where $r_{t^{'}} = r(s_{t^{'}}, a_{t^{'}})$. That is,, we push the probabilities up only by the cumulative future reward from that state. The gradient becomes

$$\nabla_\theta J(\theta) \approx \sum_{t \geqslant 0}\big(\sum_{t^{'} \geqslant t} r_{t^{'}}\big)\nabla_\theta \log \pi_\theta(s_t, a_t)$$

   We let

$$\hat{Q}_\pi(s_t, a_t) = \sum_{t^{'} \geqslant t} r_{t^{'}}$$

   be the reward to go. It is the Monte Carlo return from state s_t and action a_t. It estimates the expected reward if we take action $a_{t^{'}}$ in state $s_{t^{'}}$ (see Remark (9.3.4)). From Definition (9.3.4), the true expected reward to go, taking action a_t in state s_t, is

$$Q_\pi(s_t, a_t) = \sum_{t^{'} \geqslant t} E_{\pi_\theta}[r(s_{t^{'}}, a_{t^{'}})|s_t, a_t]$$

   so that $E_\pi[\hat{Q}_\pi(s_t, a_t)] = Q_\pi(s_t, a_t)$.

2. Use discount factor γ to ignore delayed effects

$$\nabla_\theta J(\theta) \approx \sum_{t \geqslant 0}\big(\sum_{t^{'} \geqslant t} \gamma^{t^{'}-t} r_{t^{'}}\big)\nabla_\theta \log \pi_\theta(s_t, a_t)$$

   In that case, the discounted reward to go is $\hat{Q}_\pi(s_t, a_t) = \sum_{t^{'} \geqslant t} \gamma^{t^{'}-t} r_{t^{'}}$.

However, the raw value of a trajectory is not necessarily meaningful. For instance, if rewards are all positive, we keep pushing up the probabilities of actions.

Electronic copy available at: https://ssrn.com/abstract=3307566

9.4.3.2.2 Baseline function On the other hand, we are interested in whether a reward is better or worse than expected. Thus, one solution is to introduce a baseline function dependent on the state. That is, the gradient estimator is given by

$$\nabla_\theta J(\theta) \approx \sum_{t\geqslant 0}\Big(\sum_{t'\geqslant t}\gamma^{t'-t}r_{t'} - b(s_t)\Big)\nabla_\theta \log \pi_\theta(s_t, a_t) \tag{9.4.18}$$

where $b(\cdot)$ is a baseline function. It corresponds to the gradient

$$\nabla_\theta J(\theta) = E_\pi[\sum_{t\geqslant 0}\nabla_\theta \log \pi_\theta(s_t, a_t)\big(R_t - b(s_t)\big)]$$

The baseline must be chosen carefully to reduce the variance. In general, it is a constant moving average of rewards experience so far from all the trajectories. That is, $b = \frac{1}{N}\sum_{i=1}^N R(\tau)$. This is because

$$\begin{aligned} E\big[\nabla_\theta \log \pi_\theta(\tau)b\big] &= \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)bd\tau \\ &= \int \nabla_\theta \pi_\theta(\tau)bd\tau \\ &= b\nabla_\theta \int \pi_\theta(\tau)d\tau = b\nabla_\theta 1 = 0 \end{aligned}$$

since $\pi_\theta(\tau)$ is a density. As a result, the baseline is an unbiased expectation, but with some variance. See details of the policy gradient in Algorithm (26).

Algorithm 26 Policy Gradient Algorithm

Require: : training set S, number of training steps T

```
Initialise policy parameters θ
for iteration = 1 to N do
    Sample m trajectories under the current policy
    Δθ ← 0
    for i = 1 to m do
        for t = 1 to T do,
            A_t = Σ_{t'≥t} γ^{t'-t} r_{i,t} − b
            Δθ ← Δθ + A_t ∇_θ log π_θ(s_{i,t}, a_{i,t})
        end for
        θ ← αΔθ
    end for
end for
```

9.4.4 The Actor-Critic algorithm

9.4.4.1 Description

Recall from variance reduction, we want to push up the probability of an action from a state if this action is better than the expected value of what we should get from that state. This is the Q-function and value function described in Section (9.3.3). Recall, the true expected reward to go $Q(s_t, a_t)$ from taking action a_t in state s_t is defined above and the total reward from s_t (see Equation (9.3.12)) is given by

$$V_\pi(s_t) = E_{a_t\sim\pi_\theta(s_t,a_t)}[Q_\pi(s_t, a_t)]$$

Electronic copy available at: https://ssrn.com/abstract=3307566

Formally, we want to favour an action a_t in a state s_t if the action

$$A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t)$$

is large. That is, the Advantage Function $A_\pi(s_t, a_t)$ is a centered Q-value function around it mean, the value function V. As such it reduces the variance of the gradient. Q measures the performance of each action from a given state, assuming it follows π afterwards, and A provides a measure of how each action compares the the average performance at that state. Further, we want to discard the action if that difference is small. Thus, accounting for this, we can modify the gradient estimator accordingly, getting

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \sum_{t\geqslant 0}\Big(Q_{\pi_\theta}(s_t, a_t) - V_{\pi_\theta}(s_t)\Big)\nabla_\theta \log \pi_\theta(s_t, a_t) \\ &= \sum_{t\geqslant 0} A_\pi(s_t, a_t)\nabla_\theta \log \pi_\theta(s_t, a_t)\end{aligned}$$

The better the Advantage Function $A_\pi(s_t, a_t)$, the lower the variance. This is to compare with Equation (9.4.18), which is unbiased but has high variance.
Recall from Equation (9.3.13), where T can be infinity, the Q-function seen at time t, is

$$Q_{\pi_\theta}(s_t, a_t) = r(s_t, a_t) + \gamma E_{s_{t+1}\sim p(s_{t+1}|(s_t,a_t))}[V_\pi(s_{t+1})]$$

where $\gamma \in [0, 1]$ is the discount factor. Note, in the special case where $T < \infty$ we set $\gamma = 1$. Putting terms together, the Advantage Function becomes

$$A_\pi(s_t, a_t) = r(s_t, a_t) + \gamma E_{s_{t+1}\sim p(s_{t+1}|(s_t,a_t))}[V_\pi(s_{t+1})] - V_\pi(s_t)$$

We now need to fit a model to estimate the functions Q_π, V_π, or A_π.

9.4.4.1.1 Policy evaluation We now need to perform policy evaluation. A simple approximation to the value function $V_\pi(s_t)$ (see Definition (9.3.2)) is

$$V_\pi(s_t) \approx \sum_{t'\geqslant t} r(s_{t'}, a_{t'})$$

which we denote $\widehat{V}_\pi(s_t)$. Replacing in the above expectation, the Advantage Function is approximated as

$$A_{\pi_\theta}(s_t, a_t) \approx r(s_t, a_t) + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)$$

A better approximation is to apply Monte Carlo simulation to approximate the value function as follows:

$$V_\pi(s_t) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t'\geqslant t} r(s_{i,t'}, a_{i,t'})$$

Alternatively, we can use a neural network with parameters ϕ to fit $\widehat{V}_\pi(s)$ and get the supervised regression

$$\mathcal{L}(\phi) = \frac{1}{2}\sum_i \|\widehat{V}_{\pi,\phi}(s_i) - y_i\|^2$$

where y_i is the target value. The updating rule for ϕ is

$$\Delta\phi_t \propto \nabla_\phi\big(V_{\pi,\phi}(s_t) - y_t\big)^2 \propto \big(V_{\pi,\phi}(s_t) - y_t\big)\nabla_\phi V_{\pi,\phi}(s_t)$$

Electronic copy available at: https://ssrn.com/abstract=3307566

In general, the target value is some kind of unbiased estimator of $V_\pi(s_t)$ (see above). We obtain the training data $\{(s_{i,t}, y_{i,t}); i = 1, ..., N\}$ where the target is $y_{i,t} = \sum_{t' \geqslant t} r(s_{i,t'}, a_{i,t'})$.
Another solution is to use the previously fitted value function to approximate the next value function. Given Equation (9.3.8), we get

$$\begin{aligned} y_{i,t} = \sum_{t' \geqslant t} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_{i,t}] &= r(s_{i,t}, a_{i,t}) + \gamma E_{s_{i,t+1} \sim p(s_{i,t+1}|(s_{i,t}))}[V_\pi(s_{i,t+1})] \\ &\approx r(s_{i,t}, a_{i,t}) + \gamma V_\pi(s_{i,t+1}) \\ &\approx r(s_{i,t}, a_{i,t}) + \gamma \widehat{V}_{\pi,\phi}(s_{i,t+1}) \end{aligned}$$

so that the target in the training data becomes

$$y_{i,t} = r(s_{i,t}, a_{i,t}) + \gamma \widehat{V}_{\pi,\phi}(s_{i,t+1})$$

At every step t, the target is the previously fitted value function incremented with the reward at that time. Thus, we are performing a bootstrapping to estimate the model parameters. Putting terms together, the Advantage Function is approximated as

$$A_{\pi_\theta,C}(s_{i,t}, a_{i,t}) \approx r(s_{i,t}, a_{i,t}) + \gamma \widehat{V}_{\pi,\phi}(s_{i,t+1}) - \widehat{V}_{\pi,\phi}(s_{i,t}) \tag{9.4.19}$$

We therefore need to perform the following steps:

1. Estimate $A_{\pi_\theta}(s_t, a_t)$ for current policy π.
2. Use $A_{\pi_\theta}(s_t, a_t)$ to get improved policy π'.
3. Repeat step [1].

We will show in Section (9.4.5.1) that this is similar to the policy iteration algorithm:

1. Estimate $A_{\pi_\theta}(s, a)$.
2. Set $\pi \leftarrow \pi'$.
3. Repeat step [1].

9.4.4.2 The algorithm

We can now introduce the Actor-Critic algorithm: If we do not know Q and V can we learn them? We can do it with the Q-learning. We can combine Policy Gradients and Q-learning by training both an actor (the policy) and a critic (the Q-function).
The actor is the policy deciding which action to take, and the critic tells the actor (via the value function or Q-value function) how good its action was and how it should be adjusted. Note, we can incorporate the experience replay and alleviate the task of the critic. We let the Advantage Function A_π define how much an action a is better than the expected action according to the policy π. It is given by

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

The batch actor-critic algorithm is given as follows:

1. Sample $\{s_i, a_i\}$ from $\pi_\theta(s, a)$
2. Fit $\widehat{V}_{\pi,\phi}(s)$ to sampled reward sums.

Electronic copy available at: https://ssrn.com/abstract=3307566

3. Evaluate $\widehat{A}_\pi(s_i, a_i) = r(s_i, a_i) + \gamma \widehat{V}_{\pi,\phi}(s_i^{'}) - \widehat{V}_{\pi,\phi}(s_i)$
4. Compute $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(a_i, s_i) \widehat{A}_\pi(s_i, a_i)$
5. Update $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

See details in Algorithm (27).

Algorithm 27 Actor-Critic Algorithm

Require: : training set S, number of training steps T
Initialise policy parameters θ, critic parameters ϕ
Initialise action-value function Q with random weights
for $iteration = 1$ to N **do**
  Sample m trajectories under the current policy
  $\Delta\theta \leftarrow 0$
  for $i = 1$ to m **do**
    for $t = 1$ to T **do**
      $A_t = r(s_{i,t}, a_{i,t}) + \gamma \widehat{V}_{\pi,\phi}(s_{i,t+1}) - \widehat{V}_{\pi,\phi}(s_{i,t})$
      $\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t})$
    end for
    $\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi \| A_{i,t} \|^2$
    $\theta \leftarrow \alpha \Delta\theta$
    $\phi \leftarrow \beta \Delta\phi$
  end for
end for

We can also write an online actor-critic algorithm, see Algorithm (28).

Algorithm 28 Online Actor-Critic Algorithm

Require: : training set S, number of training steps T
Initialise policy parameters θ, critic parameters ϕ
Take action $a \sim \pi_\theta(s, a)$, get $(s, a, s^{'}, r)$
Update $\widehat{V}_{\pi,\phi}$ using target $r + \gamma \widehat{V}_{\pi,\phi}(s^{'})$
Evaluate $\widehat{A}_\pi(s, a) = r(s, a) + \gamma \widehat{V}_{\pi,\phi}(s^{'}) - \widehat{V}_{\pi,\phi}(s)$
$\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(s, a) \widehat{A}_\pi(s, a)$
$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

9.4.4.3 State-dependent baselines

We can combine the policy gradient with the actor-critic to get an unbiased expectation with lower variance. In that case, the gradient estimator is given by

$$\nabla_\theta J(\theta) \approx \sum_{t \geqslant 0} \Big(\sum_{t^{'} \geqslant t} \gamma^{t^{'}-t} r_{t^{'}} - \widehat{V}_{\pi,\phi}(s_t) \Big) \nabla_\theta \log \pi_\theta(s_t, a_t) \tag{9.4.20}$$

where

$$A_{\pi_\theta, MC}(s_t, a_t) \approx \sum_{t^{'} \geqslant t} \gamma^{t^{'}-t} r_{t^{'}} - \widehat{V}_{\pi,\phi}(s_t)$$

Electronic copy available at: https://ssrn.com/abstract=3307566

In that setting the critics become state-dependent baselines. See details in Algorithm (29).

Algorithm 29 Gradient Base Actor-Critic Algorithm

Require: : training set S, number of training steps T
Initialise policy parameters θ, critic parameters ϕ
Initialise action-value function Q with random weights
for $iteration = 1$ to N **do**
  Sample m trajectories under the current policy
  $\Delta\theta \leftarrow 0$
  for $i = 1$ to m **do**
    for $t = 1$ to T **do**,
      $A_t = \sum_{t' \geqslant t} \gamma^{t'-t} r_{i,t} - \hat{V}_{\pi,\phi}(s_{i,t})$
      $\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t})$
    end for
    $\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi \|A_{i,t}\|^2$
    $\theta \leftarrow \alpha\Delta\theta$
    $\phi \leftarrow \beta\Delta\phi$
  end for
end for

Following Sutton et al. [1999], the actor-critic methods consist in

1. a policy evaluation step which learns to fit a critic $Q_{\pi,\phi}$ for the current policy π_θ, and
2. a policy improvement step which greedily optimises the policy π against the critic estimate $Q_{\pi,\phi}$.

Similarly to the approach taken with the value function, we use a neural network with parameters ϕ to fit $Q_\pi(s,a)$ and get the supervised regression

$$\mathcal{L}(\phi) = \frac{1}{2}\sum_i \|Q_{\pi,\phi}(s_i,a_i) - y_i\|^2$$

where y_i is the target value. The updating rule for ϕ is

$$\Delta\phi_t \propto \nabla_\phi \big(Q_{\pi,\phi}(s_t,a_t) - y_t\big)^2 \propto \big(Q_{\pi,\phi}(s_t,a_t) - y_t\big)\nabla_\phi Q_{\pi,\phi}(s_t,a_t)$$

In general, the target value is some kind of unbiased estimator of $Q_\pi(s_t,a_t)$. Given Equation (9.3.11), we get

$$\begin{aligned} y_{i,t} = \sum_{t' \geqslant t} E_{\pi_\theta}[r(s_{t'},a_{t'})|(s_{i,t},a_{i,t})] &= r(s_{i,t},a_{i,t}) + \gamma E_{s_{i,t+1}\sim p(s_{i,t+1}|(s_{i,t},a_{i,t}))}[Q_\pi(s_{i,t+1},a_{i,t+1})] \\ &\approx r(s_{i,t},a_{i,t}) + \gamma Q_\pi(s_{i,t+1},a_{i,t+1}) \\ &\approx r(s_{i,t},a_{i,t}) + \gamma Q_{\pi,\phi}(s_{i,t+1},a_{i,t+1}) \end{aligned}$$

so that the target in the training data becomes

$$y_{i,t} = r(s_{i,t},a_{i,t}) + \gamma Q_{\pi,\phi}(s_{i,t+1},a_{i,t+1})$$

If we do not know the action a_{t+1}, we can approximate it with its mean $\mu_\theta(s_{t+1}) = E_{\pi_\theta}[a_{t+1}]$.

Electronic copy available at: https://ssrn.com/abstract=3307566

Silver et al. [2014] proposed the deep deterministic policy gradient (DDPG) with deterministic policy $\pi_\theta(s_t, a_t) = \delta_{a_t=\mu_\theta(s_t)}$ where $\mu_\theta(s_t) = E_{\pi_\theta}[a_t]$ is the mean of the policy, and β is an arbitrary exploration distribution for a_t. Given the objective $J(\theta) = E_{s_t\sim p(s_t;\beta)}[Q_{\pi,\phi}(s_t, \mu_\theta(s_t))]$, the policy parameters are given by

$$\theta = \arg\max_\theta E_{s_t\sim p(s_t;\beta)}[Q_{\pi,\phi}(s_t, \mu_\theta(s_t))]$$

When using neural networks, full optimisation is expensive, so that stochastic gradient optimisation is used. Using the chain rule, the gradient in the policy improvement phase is given by

$$\begin{aligned}\nabla_\theta J(\theta) &\approx E_{s_t\sim p(s_t;\beta)}[\nabla_\theta Q_{\pi,\phi}(s_t, \mu_\theta(s_t))] \\ &= E_{s_t\sim p(s_t;\beta)}[\nabla_a Q_{\pi,\phi}(s_t, a)\big|_{a=\mu_\theta(s_t)} \nabla_\theta \mu_\theta(s_t)]\end{aligned}$$

While it is not prone to high variance and is trainable on off-policy data, it produces a biased policy gradient estimator making it difficult to analyse convergence and stability properties. This is to oppose to the baseline gradient in Equation (9.4.18), which provides an almost unbiased gradient, but with high variance.

Account for this result, Gu et al. [2017] proposed to use the deterministic biased estimator as a control variate for the MC policy gradient estimator, getting action-dependent baselines. It is the combination of an on-policy with an off-policy, which brings stability and efficiency. The off-policy is given by

$$\nabla_\theta J_{off}(\theta) \approx \frac{1}{N}\sum_{i=1}^N \sum_{t\geqslant 0} \nabla_\theta E_{a\sim\pi_\theta(s_{i,t},a_t)}[Q_{\pi,\phi}(s_{i,t}, \mu_\theta(s_t))]$$

which is given as above. The on-policy is given by

$$\nabla_\theta J_{on}(\theta) \approx \frac{1}{N}\sum_{i=1}^N \sum_{t\geqslant 0} \Big(\widehat{Q}_i(s_t, a_t) - \overline{Q}_{\pi,\phi}(s_{i,t}, a_{i,t})\Big)\nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t})$$

where $\overline{Q}_{\pi,\phi}(s_t, a_t)$ is a first order Taylor expansion around $a_t = \mu_\theta(s_t)$. In that case, the gradient estimator is given by

$$\nabla_\theta J(\theta) \approx \nabla_\theta J_{on}(\theta) + \nabla_\theta J_{off}(\theta) \tag{9.4.21}$$

This result holds provided that the second term can be evaluated.
To control the bias and reduce the variance, it was also suggested to combine $A_{\pi_\theta,C}$ with $A_{\pi_\theta,MC}$, getting

$$\widehat{A}_{\pi_\theta,n}(s_t, a_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r_{t'} - \widehat{V}_{\pi,\phi}(s_t) + \gamma^n \widehat{V}_{\pi,\phi}(s_{t+n})$$

where $n > 1$.
Rather than choosing a single n, Schulman et al. [2016] suggested to get a weighted combination of n-step returns called the generalised advantage estimation (GAE), given by

$$\widehat{A}_{\pi,GAE}(s_t, a_t) = \sum_{n=1}^{\infty} \omega_n \widehat{A}_{\pi_\theta,n}(s_t, a_t)$$

Letting $\omega_n \propto \lambda^{n-1}$, we get

$$\widehat{A}_{\pi,GAE}(s_t, a_t) = r(s_t, a_t) + \gamma\Big[(1-\lambda)\widehat{V}_{\pi,\phi}(s_{t+1}) + \lambda r(s_{t+1}, a_{t+1}) + \gamma\big((1-\lambda)V_{\pi,\phi}(s_{t+2}) + \lambda r(s_{t+2}, a_{t+2}) + \cdots\Big]$$

Electronic copy available at: https://ssrn.com/abstract=3307566

which we rewrite more compactly as

$$\hat{A}_{\pi,GAE}(s_t, a_t) = \sum_{t^{'} \geqslant t} (\gamma\lambda)^{t^{'}-t}\delta_{t^{'}}$$

where

$$\delta_{t^{'}} = r(s_{t^{'}}, a_{t^{'}}) + \gamma\hat{V}_{\pi,\phi}(s_{t^{'}+1}) - \hat{V}_{\pi,\phi}(s_{t^{'}})$$

and $(\gamma\lambda)$ acts as a discount rate.

9.4.5 Policy gradient improvement

9.4.5.1 Policy iteration

We present the policy iteration introduced by Kakade et al. [2002]. We start from the objective function in Equation (9.4.16) and use the decomposition in Equation (9.3.8). Recall,

$$V_{\pi_\theta}(s_t) = \sum_{t^{'} \geqslant t} E_{\pi_\theta}[\gamma^{t^{'}} r_{t^{'}}|s_t], r_{t^{'}} = r(s_{t^{'}}, a_{t^{'}})$$

and assuming the chain to be homogeneous (see Definition (14.6.27)), we get the objective

$$J(\theta) = E_{s_1 \sim p(s_1)}[V_{\pi_\theta}(s_1)]$$

Further, $E_{\tau\sim p(\tau;\theta^{'})}[\cdot]$ indicates that actions are sampled from $\pi_{\theta^{'}}$ to generate the trajectory $\tau = (s_0, a_0, s_1, a_1, ...)$. We are interested in computing the difference in policy performance $J(\theta^{'}) - J(\theta)$. The authors showed that it could be decomposed as a sum of per-timestep advantages. Starting at $t = 0$, we get

$$\begin{aligned}
J(\theta^{'}) - J(\theta) &= J(\theta^{'}) - E_{s_0\sim p(s_0)}[V_{\pi_\theta}(s_0)] \\
&= J(\theta^{'}) - E_{\tau\sim p(\tau;\theta^{'})}[V_{\pi_\theta}(s_0)] \\
&= J(\theta^{'}) - E_{\tau\sim p(\tau;\theta^{'})}[\sum_{t=0}^{\infty}\gamma^t V_{\pi_\theta}(s_t) - \sum_{t=1}^{\infty}\gamma^t V_{\pi_\theta}(s_t)] \\
&= J(\theta^{'}) + E_{\tau\sim p(\tau;\theta^{'})}[\sum_{t=0}^{\infty}\gamma^t\big(\gamma V_{\pi_\theta}(s_{t+1}) - V_{\pi_\theta}(s_t)\big)] \\
&= E_{\tau\sim p(\tau;\theta^{'})}[\sum_{t=1}^{\infty}\gamma^t r_t] + E_{\tau\sim p(\tau;\theta^{'})}[\sum_{t=0}^{\infty}\gamma^t\big(\gamma V_{\pi_\theta}(s_{t+1}) - V_{\pi_\theta}(s_t)\big)] \\
&= E_{\tau\sim p(\tau;\theta^{'})}[\sum_{t=0}^{\infty}\gamma^t\big(r_t + \gamma V_{\pi_\theta}(s_{t+1}) - V_{\pi_\theta}(s_t)\big)] \\
&= E_{\tau\sim p(\tau;\theta^{'})}[\sum_{t=0}^{\infty}\gamma^t A_{\pi_\theta}(s_t, a_t)]
\end{aligned}$$

where $\gamma^0 = 1$ and r_0 is a constant. So, we get

$$J(\theta^{'}) - J(\theta) = E_{\tau\sim p(\tau;\theta^{'})}[\sum_{t=0}^{\infty}\gamma^t A_{\pi_\theta}(s_t, a_t)]$$

Electronic copy available at: https://ssrn.com/abstract=3307566

where the expectation is under the policy $\pi_{\theta'}$ and the Advantage Function is under π_θ. We let $\rho_\pi(s)$ be the discounted visitation frequency

$$\rho_\pi(s) = \big(P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \cdots\big)$$

and re-express the difference to sum over states, getting

$$\begin{aligned} J(\theta^{'}) - J(\theta) &= \sum_{t=0}^{\infty} \sum_{s \in \mathcal{S}} P_{\pi_{\theta'}}(s_t = s) \sum_{a \in \mathcal{A}} \pi_{\theta'}(s,a) \gamma^t A_\pi(s,a) \\ &= \sum_{s \in \mathcal{S}} \sum_{t=0}^{\infty} \gamma^t P_{\pi_{\theta'}}(s_t = s) \sum_{a \in \mathcal{A}} \pi_{\theta'}(s,a) A_\pi(s,a) \\ &= \sum_{s \in \mathcal{S}} \rho_{\pi_{\theta'}}(s) \sum_{a \in \mathcal{A}} \pi_{\theta'}(s,a) A_\pi(s,a) \end{aligned}$$

Any policy such that $\sum_{a \in \mathcal{A}} \pi_{\theta'}(s,a) A_\pi(s,a) \leqslant 0$ is guaranteed to reduce J. However, there will be some states s for which the expected advantage is positive. One way around is to locally approximate J as follows:

$$L(\theta^{'}) - J(\theta) = \sum_{s \in \mathcal{S}} \rho_{\pi_\theta}(s) \sum_{a \in \mathcal{A}} \pi_{\theta'}(s,a) A_\pi(s,a)$$

where $L(\cdot)$ uses the visitation frequency ρ_{π_θ}. We will show that L matches J to the first order. To do so we work with expectation. Expending $\tau \sim p(\tau; \theta^{'})$ and using importance sampling (see Appendix (15.2.2.4)) to modify the distribution, we get

$$\begin{aligned} E_{\tau \sim p(\tau;\theta')}[\sum_{t=0}^{\infty} \gamma^t A_{\pi_\theta}(s_t, a_t)] &= \sum_{t=0}^{\infty} E_{s_t \sim p(s_t;\theta')}[E_{a_t \sim \pi_{\theta'}(a_t,s_t)}[\gamma^t A_{\pi_\theta}(s_t, a_t)]] \\ &= \sum_{t=0}^{\infty} E_{s_t \sim p(s_t;\theta')}[E_{a_t \sim \pi_\theta(a_t,s_t)}[\frac{\pi_{\theta'}(a_t, s_t)}{\pi_\theta(a_t, s_t)} \gamma^t A_{\pi_\theta}(s_t, a_t)]] \end{aligned}$$

Assuming that $p(s_t; \theta)$ is close to $p(s_t; \theta^{'})$ when π_θ is close to $\pi_{\theta'}$, one can approximate the density $p(s_t; \theta^{'})$ in the outer expectation with $p(s_t; \theta)$, getting

$$\sum_{t=0}^{\infty} E_{s_t \sim p(s_t;\theta')}[E_{a_t \sim \pi_\theta(a_t,s_t)}[\frac{\pi_{\theta'}(a_t, s_t)}{\pi_\theta(a_t, s_t)} \gamma^t A_{\pi_\theta}(s_t, a_t)]] \approx \sum_{t=0}^{\infty} E_{s_t \sim p(s_t;\theta)}[E_{a_t \sim \pi_\theta(a_t,s_t)}[\frac{\pi_{\theta'}(a_t, s_t)}{\pi_\theta(a_t, s_t)} \gamma^t A_{\pi_\theta}(s_t, a_t)]]$$

We let the RHS be denoted by $\overline{A}(\theta^{'})$, and then get

$$J(\theta^{'}) - J(\theta) \approx \overline{A}(\theta^{'}) \Longrightarrow \theta^{'} \leftarrow \arg\max_{\theta'} \overline{A}(\theta^{'}) \tag{9.4.22}$$

In the special case where $\theta^{'} = \theta$, we get

$$\overline{A}(\theta) = E_{\tau \sim p(\tau;\theta)}[\sum_{t=0}^{\infty} \gamma^t A_{\pi_\theta}(s_t, a_t)]$$

We now want to find some bounds to the distribution change. Kakade et al. [2002] obtained some bounds in the special case of mixture of policies

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\pi_{new}(s,a) = (1-\alpha)\pi_{old}(s,a) + \alpha\pi^{'}(s,a)$$

Schulman et al. [2015] extended the result to general stochastic policies by replacing α with a distance measure between π and $\pi^{'}$.

***Claim* 1** *We claim that if π_θ is an arbitrary distribution, then $\pi_{\theta'}$ is close to π_θ if $|\pi_{\theta'}(a_t,s_t) - \pi_\theta(a_t,s_t)| \leqslant \epsilon$ for all s_t.*

Following Lemma (14.6.7), we can infer that $\pi_{\theta'}(a_t,s_t)$ takes a different action than $\pi_\theta(a_t,s_t)$ with probability at most ϵ. We write the mass function $p(s_t;\theta^{'})$ as

$$p(s_t;\theta^{'}) = (1-\epsilon)^t p(s_t;\theta) + \big(1-(1-\epsilon)^t\big)p(s_t;\xi)$$

where $p(s_t;\xi)$ is another distribution, possibly from a mistake. Using the identity $(1-\epsilon)^t \geqslant 1-\epsilon t$ for $\epsilon \in [0,1]$, we get

$$\begin{aligned} |p(s_t;\theta^{'}) - p(s_t;\theta)| = \big(1-(1-\epsilon)^t\big)|p(s_t;\xi) - p(s_t;\theta)| &\leqslant 2\big(1-(1-\epsilon)^t\big) \\ &\leqslant 2\epsilon t \end{aligned}$$

Given a smooth function $f(\cdot)$, we can therefore bound its expectation as

$$\begin{aligned} E_{p(s_t;\theta')}[f(s_t)] = \sum_{s_t} p(s_t;\theta^{'})f(s_t) &\geqslant \sum_{s_t} p(s_t;\theta)f(s_t) - |p(s_t;\theta^{'}) - p(s_t;\theta)| \max_{s_t} f(s_t) \\ &\geqslant E_{p(s_t;\theta)}[f(s_t)] - 2\epsilon t \max_{s_t} f(s_t) \end{aligned}$$

Back to $J(\theta^{'}) - J(\theta)$, we get

$$\begin{aligned} &\sum_{t=0}^{\infty} E_{s_t\sim p(s_t;\theta')}[E_{a_t\sim\pi_\theta(a_t,s_t)}[\frac{\pi_{\theta'}(a_t,s_t)}{\pi_\theta(a_t,s_t)}\gamma^t A_{\pi_\theta}(s_t,a_t)]] \geqslant \\ &\sum_{t=0}^{\infty} E_{s_t\sim p(s_t;\theta)}[E_{a_t\sim\pi_\theta(a_t,s_t)}[\frac{\pi_{\theta'}(a_t,s_t)}{\pi_\theta(a_t,s_t)}\gamma^t A_{\pi_\theta}(s_t,a_t)]] - \sum_{t=0}^{\infty} 2\epsilon t C \end{aligned}$$

where C is $O(\frac{r_{\max}}{1-\gamma})$. Simplifying notation we get the bound $\overline{A}(\theta^{'}) - \sum_{t=0}^{\infty} 2\epsilon t C$.
Schulman et al. [2015] got a better bound by using the Kullback-Leibler divergence $D_{K,L}(P||Q) = \sum_y P(y)\log\frac{P(y)}{Q(y)}$. That is, $D_{K,L}(p_1(x)||p_2(x)) = E_{x\sim p_1(x)}[\log\frac{p_1(x)}{p_2(x)}]$. Identifying P to $\pi_{\theta'}(a_t,s_t)$ and Q to $\pi_\theta(a_t,s_t)$, we get

$$|\pi_{\theta'}(a_t,s_t) - \pi_\theta(a_t,s_t)| \leqslant \sqrt{\frac{1}{2}D_{K,L}(\pi_{\theta'}(a_t,s_t)||\pi_\theta(a_t,s_t))}$$

where $D_{K,L}(\pi_{\theta'}(a_t,s_t)||\pi_\theta(a_t,s_t))$ bounds the state marginal difference.

9.4.5.2 An optimisation problem

Applying a constraint on the KL divergence between the new policy and the old one defines the trust region constraint (TRC). Given the relation in Equation (9.4.22), we need to solve the constrained optimisation problem

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\theta^{'} \leftarrow \arg\max_{\theta^{'}} \overline{A}(\theta^{'})$$
$$\text{such that } D_{K,L}(\pi_{\theta^{'}}(a_t, s_t)||\pi_\theta(a_t, s_t)) \leqslant \epsilon$$

Thus, the KL divergence is bounded at every point in the state space. For small enough ϵ we are guaranteed to improve the difference $J(\theta^{'}) - J(\theta)$.
One solution to this problem is to follow the Lagrange approach presented in Section (10.4). The Lagrangian is given by

$$L(\theta^{'}, \lambda) = \overline{A}(\theta^{'}) - \lambda\Big(D_{K,L}(\pi_{\theta^{'}}(a_t, s_t)||\pi_\theta(a_t, s_t)) - \epsilon\Big)$$

The steps are as follows:

1. Maximise $L(\theta^{'}, \lambda)$ with respect to $\theta^{'}$.
2. $\lambda \leftarrow \lambda + \alpha\big(D_{K,L}(\pi_{\theta^{'}}(a_t, s_t)||\pi_\theta(a_t, s_t)) - \epsilon\big)$.

9.4.5.2.1 Recovering the natural policy gradient Alternatively, we can perform a first order Taylor expansion (linearisation) of $\overline{A}(\theta^{'})$ around $\overline{A}(\theta)$, getting

$$\theta^{'} \leftarrow \arg\max_{\theta^{'}} \nabla_\theta \overline{A}(\theta)^\top \Delta\theta$$

where $\Delta\theta = \theta^{'} - \theta$.
Using Equation (9.4.17), we get the gradient

$$\nabla_{\theta^{'}} \overline{A}(\theta^{'}) = \sum_{t=0}^{\infty} E_{s_t \sim p(s_t;\theta)}[E_{a_t \sim \pi_\theta(a_t,s_t)}[\frac{\pi_{\theta^{'}}(a_t, s_t)}{\pi_\theta(a_t, s_t)} \gamma^t \nabla_{\theta^{'}} \log \pi_{\theta^{'}}(a_t, s_t) A_{\pi_\theta}(s_t, a_t)]]$$

In the special case where $\theta^{'} = \theta$, it simplifies to

$$\nabla_\theta \overline{A}(\theta) = \sum_{t=0}^{\infty} E_{s_t \sim p(s_t;\theta)}[E_{a_t \sim \pi_\theta(a_t,s_t)}[\gamma^t \nabla_\theta \log \pi_\theta(a_t, s_t) A_{\pi_\theta}(s_t, a_t)]] = \nabla_\theta J(\theta)$$

which is the normal policy gradient.
So logically, we can wonder if you can just use the gradient. One solution could be to use the gradient ascent as follows:

$$\theta^{'} \leftarrow \arg\max_{\theta^{'}} \nabla_\theta J^\top(\theta) \Delta\theta$$
$$\text{such that } \|\Delta\theta\|^2 \leqslant \epsilon$$

where the update is

$$\theta^{'} = \theta + \frac{\epsilon}{\|\nabla_\theta J(\theta)\|^2} \nabla_\theta J(\theta)$$

Another solution would be to rescale the gradient and consider the problem

$$\theta^{'} \leftarrow \arg\max_{\theta^{'}} \nabla_\theta J^\top(\theta) \Delta\theta$$
$$\text{such that } D_{K,L}(\pi_{\theta^{'}}(a_t, s_t)||\pi_\theta(a_t, s_t)) \leqslant \epsilon$$

Electronic copy available at: https://ssrn.com/abstract=3307566

Even though the constraints are not the same, we can perform a second order Taylor expansion of the Kullback-Leibler divergence, getting

$$D_{KL}(\pi_{\theta'}||\pi_\theta) \approx \frac{1}{2}(\Delta\theta)^\top F \Delta\theta$$

where F is the Fisher-information matrix

$$E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\big(\nabla_\theta \log \pi_\theta(s,a)\big)^\top]$$

which can be statistically estimated. Then we get the update

$$\theta^{'} = \theta + \alpha F^{-1}\nabla_\theta J(\theta)$$

We can solve for α while solving $F^{-1}\nabla_\theta J(\theta)$ (see Schulman et al. [2015]). We get

$$\alpha = \frac{2\epsilon}{\big(\nabla_\theta J(\theta)\big)^\top F \nabla_\theta J(\theta)}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

Part IV

Optimisation

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 10

Introduction to constrained optimisation

In mathematics and computer science, an optimisation problem is the problem of finding the best solution from all feasible solutions. Optimisation problems can be divided into two categories depending on whether the variables are (1) continuous or (2) discrete. In an optimisation problem with discrete variables (discrete optimisation), we are looking for an object such as an integer, permutation or graph from a finite (or possibly countably infinite) set. Problems with continuous variables include constrained problems and multimodal problems. In this chapter we are going to focus on the latter. In such problems, we try to optimise (maximise or minimise) some quantity, while satisfying some constraints. Optimisation problems are of particular interest in the area of Operations Research and they appear in many real situations. It has important applications in several fields, including artificial intelligence, machine learning, auction theory, and software engineering.

See textbooks by Bertsekas [1996] [2015], Bazaraa et al. [1993], Nocedal et al. [1999], Boyd et al. [2004], Sun et al. [1999], among other.

10.1 Introduction

10.1.1 Defining the problem

10.1.1.1 The constrained problem

We consider a system with the real-valued properties

$$g_m \text{ for } m = 0, .., P - 1$$

making the objectives of the system to be optimised. Given a N-dimensional vector of real-valued parameter $X \subset \mathbb{R}^N$ the optimisation problem can always be written as

$$\min f_m(X)$$

where $f_m(\cdot)$ is a function by which g_m is calculated and where each element $X(i)$ of the vector is bounded by lower and upper limits $L_i \leqslant X(i) \leqslant U_i$ which define the search space $\mathcal{S}$. Depending on the linearity or non-linearity of the objective function we can distinguish between linear and non-linear optimisation.
We follow Lueder [1990] who showed that all functions $f_m(\cdot)$ can be combined in a single objective function $H : X \subset \mathbb{R}^N \to \mathbb{R}$ expressed as the weighted sum

$$H(X) = \sum_{m=1}^{P} w_m f_m(X)$$

Electronic copy available at: https://ssrn.com/abstract=3307566

where the weighting factors w_m define the importance of each objective of the system. Hence, the optimisation problem becomes

$$\min H(X)$$

so that all the local and global minima (when the region of eligibility in X is convex) can be found. However, most problems involves a single objective function, so that the optimisation function simplifies. Most complex search problems such as optimisation problems are constrained numerical problem (CNOP) more commonly called general nonlinear programming problems with constraints given by

$$g_i(X) \leqslant 0 \, , i = 1, .., p$$
$$h_j(X) = 0 \, , j = 1, .., q$$

Equality constraints are usually transformed into inequality constraints by

$$|h_j(X)| - \epsilon \leqslant 0$$

where ϵ is the tolerance allowed. Given the search space $\mathcal{S} \subset \mathbb{R}^N$, we let $\mathcal{F}$ be the set of all solutions satisfying the constraints of the problems called the feasible region. It is defined by the intersection of $\mathcal{S}$ and the set of $p + q$ additional constraints. At any point $X \in \mathcal{F}$, the constraints $g_i(\cdot)$ that satisfy $g_i(X) = 0$ are active constraints at X while equality constraints $h_j(\cdot)$ are active at all points of $\mathcal{F}$. Many practical problems have objective functions that are non-differentiable, non-continuous, non-linear, noisy, multi-dimensional and have many local minima.

10.1.1.2 Some examples

We let $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$, $h : \mathbb{R}^n \to \mathbb{R}^l$ and consider the optimisation problem

$$\min_x f(x) \text{ such that } g(x) \leqslant 0 \, , h(x) = 0 \tag{10.1.1}$$

As discussed above, any equality constraint $h_i(x)$ can be transformed into two inequality constraints

$$h_i(x) \geqslant 0 \text{ and } -h_i(x) \geqslant 0$$

such that

$$\min_x f(x) \text{ such that } g(x) \leqslant 0 \tag{10.1.2}$$

is sufficiently general.
Some examples of optimisation problems are

- Linear regression:

$$\min_w \|xw - y\|^2$$

- Classification (logistic regression or SVM)

$$\min_w \sum_{i=1}^n \log\left(1 + e^{-y_i x_i^\top w}\right)$$

or

$$\|w\|^2 + C \sum_{i=1}^n \xi_i \text{ such that } \xi_i \geqslant 1 - y_i x_i^\top w \, , \xi_i \geqslant 0$$

Electronic copy available at: https://ssrn.com/abstract=3307566

- Maximum likelihood estimation:

$$\max_{\theta} \sum_{i=1}^{n} \log p_\theta(x_i)$$

- k-,means

$$\min_{\mu_1,\dots,\mu_k} J(\mu) = \sum_{j=1}^{k} \sum_{i \in C_j} \|x_i - \mu_j\|^2$$

Some important examples in machine learning are

- SVM loss:

$$f(w) = \big(1 - y_i x_i^\top w\big)^+$$

- Binary logistic loss:

$$f(w) = \log\big(1 + e^{-y_i x_i^\top w}\big)$$

10.1.2 Conditions for a local optimum

10.1.2.1 Local minimum

Necessary and sufficient conditions for a local optimum: x^* is a local minimum of $f(x)$ if and only if

1. f has a zero gradient at x^*:

$$\nabla_x f(x^*) = 0$$

2. and the Hessian of f at x^* is positive semi-definite:

$$v^\top \big(\nabla^2 f(x^*)\big) v \geqslant 0 \ , \forall v \in \mathbb{R}^n$$

where

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{pmatrix}$$

10.1.2.2 Local maximum

Similarly, x^* is a local maximum of $f(x)$ if and only if

1. f has a zero gradient at x^*:

$$\nabla_x f(x^*) = 0$$

2. and the Hessian of f at x^* is negative semi-definite:

$$v^\top \big(\nabla^2 f(x^*)\big) v \leqslant 0 \ , \forall v \in \mathbb{R}^n$$

10.1.3 Conditions for constrained local optimum

We explore the condition to decrease the cost function in the case of both an equality constraint and an inequality constraint. We use the notion of steepest descent described in Appendix (10.2.1.1).

Electronic copy available at: https://ssrn.com/abstract=3307566

10.1.3.1 The equality constraint

At any point x the direction of the steepest descent of the cost function $f(x)$ is given by $\Delta x = -\nabla_x f(x)$. Thus, to move δx from x such that $f(x + \delta x) < f(x)$ we must have

$$\delta x \cdot \big(-\nabla_x f(x)\big) > 0$$

We then consider the conditions to remain on the constraint surface: Normals to the constraint surface are given by $\nabla_x h(x)$. Note, the direction of the normal is arbitrary as the constraint can be imposed as either $h(x) = 0$ or $-h(x) = 0$. To move a small δx from x and remain on the constraint surface we need to move in a direction which is orthogonal to $\nabla_x h(x)$.
More generally, for x_F a feasible point, we consider the case

$$\nabla_x f(x_F) = \nu \nabla_x h(x_F)$$

where ν is a scalar. It occurs if δx is orthogonal to $\nabla_x h(x_F)$, so that

$$\delta x \cdot \big(-\nabla_x f(x_F)\big) = -\delta x \cdot \nu \nabla_x h(x_F) = 0$$

So a constrained local optimum occurs at x^* when $\nabla_x f(x^*)$ and $\nabla_x h(x^*)$ are parallel, that is,

$$\nabla_x f(x^*) = \nu \nabla_x h(x^*)$$

10.1.3.2 The inequality constraint

Again, x_F denotes a feasible point. The necessary and sufficient conditions for a constrained local minimum are the same as for an unconstrained local minimum. However, we observe two cases

1. the constraint is not active at the local minimum ($g(x^*) < 0$): therefore the local minimum is identified by the same conditions as in the unconstrained case.
2. the constrained local minimum occurs on the surface of the constraint surface: thus, we have an optimisation problem with an equality constraint ($g(x) = 0$). A local optimum occurs when $\nabla_x f(x)$ and $\nabla_x g(x)$ are parallel

   $$-\nabla_x f(x) = \lambda \nabla_x g(x)$$

   However, for some points, we do not have a local minimum since $-\nabla_x f(x_F)$ points in towards the feasible region. Thus, the constrained local minimum occurs when $-\nabla_x f(x)$ and $\Delta_x g(x)$ point in the same direction

   $$-\nabla_x f(x) = \lambda \nabla_x g(x) \text{ and } \lambda > 0$$

To summarise, if x^* corresponds to a constrained local minimum, then

1. unconstrained local minimum occurs in the feasible region
   (a) $g(x^*) < 0$
   (b) $\nabla_x f(x^*) = 0$
   (c) $\nabla_{xx} f(x^*)$ is a positive semi-definite matrix
2. unconstrained local minimum lies outside the feasible region
   (a) $g(x^*) = 0$
   (b) $-\nabla_x f(x^*) = \lambda \nabla_x g(x^*)$ with $\lambda > 0$
   (c) $v^\top \nabla_{xx} L(x^*) v \geqslant 0$ for all v orthogonal to $\nabla_x g(x^*)$

Electronic copy available at: https://ssrn.com/abstract=3307566

10.1.4 Conditions for a global optimum

Convexity plays a key role in mathematical programming. Convex programs minimise convex optimisation functions subject to convex constraints ensuring that every local minimum is always a global minimum. The second order convexity conditions are defined as follows:

Theorem 10.1.1 *Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is twice differentiable. Then f is convex if and only if for all $x \in \text{dom}\, f$,*

$$\nabla^2 f(x) \geqslant 0$$

In general, we want the optimisation problem to be convex. We define a convex optimisation problem as follow:

Definition 10.1.1 *An optimisation problem is convex if its objective is a convex function, the inequality constraint g is convex and the equality constraint is affine.*

$$\min_x f(x) \text{ (convex function) such that } g(x) \leqslant 0 \text{ (convex set) }, h(x) = 0 \text{ (affine)}$$

Theorem 10.1.2 *If $\widehat{x}$ is a local minimiser of a convex optimisation problem, it is a global minimiser.*

The following theorem emphasises the reasons why we want a convex problem. It applies to smooth functions.

Theorem 10.1.3 $\nabla f(x) = 0$ *if and only if x is a global minimiser of $f(x)$.*

Proof: For $\nabla f(x) = 0$ we have

$$f(y) \geqslant f(x) + \nabla f^\top(x)(y - x) = f(x)$$

For $\nabla f(x) \neq 0$ there is a direction of descent.

10.1.5 Some solutions

When dealing with constrained optimisation problem we want to transform the constraint problem to one of the following:

- a series of unconstrained problems
- a single but larger unconstrained problem
- another simpler constraint problem (dual, convex)

Some of the existing solutions are

1. Penalty and barriers
    - Associate a (adaptive) penalty cost with violation of the constraint
    - Associate an additional force compensating the gradient into the constraint (augmented Lagrangian)
    - Associate a log barrier with a constraint, becoming ∞ for violation (interior point method)
2. Gradient projection methods (mostly for linear constraints)
    - For active constraints, project the step direction to become tangantial
    - When checking a step, always pull it back to the feasible region
3. Lagrangian and dual methods

Electronic copy available at: https://ssrn.com/abstract=3307566

- Rewrite the constrained problem into an unconstrained one
- Or rewrite it as a (convex) dual problem

4. Simplex methods (linear constraints)
   - Walk along the constraint boundaries

10.2 Unconstrained optimisation

10.2.1 Some classical methods

10.2.1.1 The gradient descent

One of the simplest approach for solving convex unconstrained optimisation problem is the Gradient Descent (GD). The gradient descent search determines a vector x minimising the function $f(x)$ by starting with an arbitrary initial vector, and then repeatedly modifying it in small steps. At each step, the vector is altered in the direction that produces the steepest descent along the error surface, until a global minimum error is reached. This direction is found by computing the derivative of f with respect to each component of the vector x. It is called the gradient of f with respect to x, given by

$$\nabla f(x) = [\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n}]$$

***Remark* 10.2.1** *The (ascent) gradient specifies the direction that produces the steepest increase in f, while its negative produces the steepest decrease.*

The Gradient Descent is described in Algorithm (30).

Algorithm 30 Gradient Descent

1: **for** $t = 1$ **to** T **do**
2: $\quad x_{t+1} \leftarrow x_t + \Delta x_t$ where $\Delta x_t = -\eta_t \nabla f(x_t)$
3: **end for**

The update is given by

$$x_{t+1} = x_t - \eta_t \nabla f(x_t)$$

In that setting t is the time step and η_t is the step size or learning rate. There are several ways for choosing the step size such as:

- Exact line search
$$\eta_t = \arg\min_{\eta} f\big(x - \eta \nabla f(x)\big)$$
- Backtracking line search. Let $\alpha \in (0, \frac{1}{2})$, $\beta \in (0, 1)$. Multiply $\eta = \beta\eta$ until
$$f\big(x - \eta \nabla f(x)\big) \leqslant f(x) - \alpha\eta \|\nabla f(x)\|^2$$

Electronic copy available at: https://ssrn.com/abstract=3307566

10.2.1.2 The subgradient descent

We can simplify the Gradient Descent method by using the Subgradient method (see Appendix (14.2.1.2)), which are iterative methods for solving convex minimisation problems (see Shor [1985]). They are convergent even when applied to a non-differentiable objective function. When the objective function is differentiable, sub-gradient methods for unconstrained problems use the same search direction as the method of steepest descent. See Algorithm (31).

Algorithm 31 Subgradient Descent

1: **for** $t = 1$ **to** T **do**
2: $\quad x_{t+1} \leftarrow x_t - \eta_t \psi_t$
3: **end for**

where η_t is the step size and $\psi_t \in \partial f(x_t)$ denotes a subgradient of f at x_t and x_t is the t-th iterate of x. If f is differentiable, then its only subgradient is the gradient vector ∇f itself. It may happen that ψ_t is not a descent direction for f at x_t. We therefore maintain a list of f_{best} that keeps track of the lowest objective function value found so far. It is given by

$$f_{best,t} = \{f_{best,t-1}, f(x_t)\}$$

Many different types of step-size rules are used by subgradient methods. We consider methods with proof of convergence, where the step-sizes are determined off-line, before the method is iterated such as

- Constant step size: $\eta_t = \eta$
- Constant step length: $\eta_t = \frac{\gamma}{\|\psi_t\|_2}$ which gives $\|x_{t+1} - x_t\|_2 = \gamma$.
- Square summable but not summable step size, that is, any step sizes satisfying

$$\eta_t \geqslant 0\,,\ \sum_{t=1}^{\infty} \eta_t^2 \leqslant \infty\,,\ \sum_{t=1}^{\infty} \eta_t = \infty$$

- Nonsummable diminishing, that is, any step sizes satisfying

$$\eta_t \geqslant 0\,,\ \lim_{t\to\infty} \eta_t = 0\,,\ \sum_{t=1}^{\infty} \eta_t = \infty$$

Some examples of non-differentiable convex functions used in machine learning are

$$f(x) = \left(1 - a^\top x\right)^+\,,\ f(x) = \|x\|_1\,,\ f(x) = \sum_{r=1}^{k} \sigma_r(x)$$

where σ_r is the rth singular value of x.
Shor proved that for constant step-length and scaled subgradients having Euclidean norm equal to one, the subgradient method converges to an arbitrarily close approximation to the minimum value, that is,

$$\lim_{k\to\infty} f_{best,k} - f^* < \epsilon$$

Electronic copy available at: https://ssrn.com/abstract=3307566

10.2.1.3 The Newton's method

The idea behind the Newton's method is to use a second-order approximation to the function

$$f(x+\Delta x) \approx f(x) + \nabla f^\top(x)\Delta x + \frac{1}{2}\Delta x^\top \nabla^2 f(x)\Delta x$$

Choose Δx to minimise the above equation, we get

$$\Delta x = -\big(\nabla^2 f(x)\big)^{-1}\Delta f(x)$$

We get the descent direction

$$\nabla f^\top(x)\Delta x = -\nabla f^\top(x)\big(\nabla^2 f(x)\big)^{-1}\Delta f(x) < 0$$

10.2.1.4 The convergence

The convergence rate is as follows:

- Strongly convex case: $\nabla^2 f(x) \geqslant mI$, then we have Linear convergence. For some $\gamma \in (0,1)$, $f(x_t) - f(x^*) \leqslant \gamma^t$, $\gamma < 1$. We get
$$f(x_t) - f(x^*) \leqslant \gamma^t \text{ or } t \geqslant \frac{1}{\gamma}\log\frac{1}{\epsilon} \Rightarrow f(x_t) - f(x^*) \leqslant \epsilon$$
- Smooth case: $\|\nabla f(x) - \nabla f(y)\| \leqslant C\|x-y\|$. We get
$$f(x_t) - f(x^*) \leqslant \frac{K}{t^2}$$
- Newton's method is often faster, especially when f has long valleys.

Note, inverting a Hessian is very expensive $O(d^3)$. One can approximate the inverse Hessian with BFGS or Limited-memory BFGS. We can also use the Conjugate Gradient Descent. For unconstrained non-convex problems, theses methods will find local optima.

10.2.2 Adding constraints

10.2.2.1 The projected subgradient descent

One extension is the projected subgradient descent (PSD) which solves a constrained optimisation problem (see Lemarechal [2001]). Subgradient-projection methods are suitable for convex minimisation problems with very large number of dimensions, since they require little storage. Given a convex constraint set X we want to minimise $\min_{x\in X} f(x)$. We do subgradient steps and project x_t back into X at every iteration

$$x_{t+1} = \prod_X \big(x_t - \eta\psi_t\big)$$

where $\prod_X$ is a projection on X and ψ_t is any subgradient of f at x_t. Sketch of proof:

$$\|\prod_X (x_t) - x^*\| \leqslant \|x_t - x^*\|$$

if $x^* \in X$.
Any decreasing, non-summable step size $\eta_t \to 0$, $\sum_{t=1}^{\infty}\eta_t = \infty$ gives

Electronic copy available at: https://ssrn.com/abstract=3307566

$$f(x_{avg(t)}) - f(x^*) \to 0$$

Further, with $\eta_t \propto \frac{1}{\sqrt{t}}$, we get

$$f(x_{avg(t)}) - f(x^*) \leqslant \frac{C}{\sqrt{t}}$$

10.2.2.2 The subgradient descent

The subgradient method can be extended to solve the inequality constrained problem

$$\min f(x) \text{ such that } g_i(x) \leqslant 0\ , i = 1, ..., m$$

where $g_i(\cdot)$, $i = 1, ..., m$, are convex functions. The algorithm takes the same form as the unconstrained case in Algorithm (31). However, ψ_t is a subgradient of the objective or one of the constraint functions at x. That is,

$$\psi_t = \begin{cases} \partial f(x) \text{ if } g_i(x) \leqslant 0 \forall i = 1, ..., m \\ \partial g_j(x) \text{ for some } j \text{ such that } g_j(x) > 0 \end{cases}$$

where ∂f denotes the subdifferential of f. If the current point is feasible, the algorithm uses an objective subgradient, while if the current point is infeasible, the algorithm chooses a subgradient of any violated constraint.

10.2.2.3 The Newton method

The simpler case is Linear constraints $Ax = b$. For example, in the case of Newton method, we consider $f(x + \Delta x) - f(x)$ and minimise

$$\min_{\Delta x} \nabla f^\top(x)\Delta x + \frac{1}{2}\Delta x^\top \nabla^2 f(x)\Delta x \text{ such that } A\Delta x = 0$$

The solution Δx satisfies

$$A(x + \Delta x) = Ax + A\Delta x = b$$

In the case of inequality constraints we get

$$\min_{\Delta x} \nabla f^\top(x)\Delta x + \frac{1}{2}\Delta x^\top \nabla^2 f(x)\Delta x \text{ such that } g_i(x + \Delta x) \leqslant 0$$

10.3 A first approach to constrained optimisation

Since equality constraints can be expressed in terms of inequality constraints, we will focus on the latter.

10.3.1 Penalty and barriers

We use the following convention

- A barrier is ∞ for $g(x) > 0$
- A penalty is zero for $g(x) \leqslant 0$ and increases with $g(x) > 0$

Electronic copy available at: https://ssrn.com/abstract=3307566

We let $I(a) = 1$ if $a > 0$, 0 otherwise. We let $I_-(a) = \infty$ if $a > 0$, 0 otherwise. That is, $I_-(a) = \infty I(a)$. Further, we let $I_0(a) = \infty$ unless $a = 0$.
We present in Appendix (10.6) a generalisation of the penalty function approach. The penalty function has the general form

$$P(g(x)) = \Psi(g(x))$$

where the function $\Psi(x)$ needs to be defined. We are now going to discuss some of these functions.

10.3.1.1 Log barrier method

In the logarithmic barrier method, rather than considering the problem in Equation (10.1.1), we rewrite the problem as

$$\min_x f(x) + \sum_{i=1}^{m} I_-(g_i(x))$$

where $I_-(a) = \infty$ if $a > 0$, 0 otherwise. That is, $I_-(a) = \infty I(a)$. Note, we get the approximation $I_-(a) \approx -\mu \log(-a)$ for small μ. Thus, we can rewrite the minimisation problem as

$$\min_x f(x) - \mu \sum_{i=1}^{m} \log(-g_i(x))$$

Some of the consequences of this re-formulation are:

- For $\mu \to 0$ then $-\mu \log(-g)$ converges to $\infty I(g > 0) = I_-(g > 0)$
- The barrier gradient $\nabla - \log(-g) = \frac{\nabla g}{g}$ pushes away from the constraint
- Eventually we want to have a very small μ, but doing so makes the barrier very non-smooth, which is bad for gradient and 2nd order methods

10.3.1.2 Central path

In this approach, every μ defines a different optimal $x^*(\mu)$ given by

$$x^*(\mu) = \arg\min_x f(x) - \mu \sum_{i=1}^{m} \log(-g_i(x))$$

As a result, each point on the path can be understood as the optimal compromise of minimising $f(x)$ and a repelling force of the constraints (which corresponds to the dual variables $\lambda^*(\mu)$). Detail is given in the Algorithm (32).

Algorithm 32 Log Barrier method

Require: Input: initial $x \in \mathbb{R}^n$, function $f(x)$, $g(x)$, $\nabla f(x)$, $\nabla g(x)$, tolerances θ, ϵ
Require: Output: x
Require: Initialise $\mu = 1$
1: **repeat**
2: find $x \leftarrow \arg\min_x f(x) - \mu \sum_{i=1}^{m} \log(-g_i(x))$ with tolerance 10θ
3: decrease $\mu \leftarrow \frac{\mu}{10}$
4: **until** $|\Delta x| < \theta$ and $\forall i : g_i(x) < \epsilon$

Electronic copy available at: https://ssrn.com/abstract=3307566

10.3.1.3 Squared penalty method

In the Squared penalty method, rather than considering the problem in Equation (10.1.1), we rewrite the problem as

$$\min_x f(x) + \mu \sum_{i=1}^{m} I(g_i(x) > 0) g_i^2(x)$$

Note, this method will always lead to some violation of constraints. A better idea would be to add an out-pushing gradient/force $-\nabla g_i(x)$ for every constraint $g_i(x) > 0$ that is violated. Ideally, the out-pushing gradient mixes with $-\nabla f(x)$ exactly such that the result becomes tangential to the constraint. This idea leads to the augmented Lagrangian approach. Detail is given in the Algorithm (33).

Algorithm 33 Squared Penalty method

Require: Input: initial $x \in \mathbb{R}^n$, function $f(x)$, $g(x)$, $\nabla f(x)$, $\nabla g(x)$, tolerances θ, ϵ
Require: Output: x
Require: Initialise $\mu = 1$
1: **repeat**
2: find $x \leftarrow \arg\min_x f(x) + \mu \sum_{i=1}^{m} I(g_i(x) > 0) g_i^2(x)$ with tolerance 10θ
3: decrease $\mu \leftarrow \frac{\mu}{10}$
4: **until** $|\Delta x| < \theta$ and $\forall i : g_i(x) < \epsilon$

10.3.2 The augmented Lagrangian approach

10.3.2.1 The equality constraint

We first consider an equality constraint in Equation (10.1.1) before addressing inequalities which we rewrite as

$$\min_x f(x) + \mu \sum_{i=1}^{m} h_i^2(x) + \sum_{i=1}^{m} \lambda_i h_i(x)$$

Some of the consequences of this re-formulation are:

- The gradient $\nabla h_i(x)$ is always orthogonal to the constraint
- By tuning λ_i we can induce a virtual gradient $\lambda_i \nabla h_i(x)$
- The term $\mu \sum_{i=1}^{m} h_i^2(x)$ penalises the function as before

We first minimise the above equation for some μ and λ_i, which will lead to a (slight) penalty $\mu \sum_{i=1}^{m} h_i^2(x)$. For the next iteration, we choose λ_i to generate exactly the gradient that was previously generated by the penalty. The optimality condition after an iteration is

$$x^{'} = \arg\min_x f(x) + \mu \sum_{i=1}^{m} h_i^2(x) + \sum_{i=1}^{m} \lambda_i h_i(x)$$

which gives

$$0 = \nabla f(x^{'}) + \mu \sum_{i=1}^{m} 2h_i(x^{'}) \nabla h_i(x^{'}) + \sum_{i=1}^{m} \lambda_i \nabla h_i(x^{'})$$

We then update the λ for the next iteration

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}\sum_{i=1}\lambda_{i,new}\nabla h_i(x^{'}) &= \mu\sum_{i=1}^{m}2h_i(x^{'})\nabla h_i(x^{'})+\sum_{i=1}\lambda_{i,old}\nabla h_i(x^{'})\\ \lambda_{i,new} &= \lambda_{i,old}+2\mu h_i(x^{'})\end{aligned}$$

Detail is given in the Algorithm (34). As a consequence

- We do not have to take the penalty limit $\mu\to\infty$ but still can have exact constraints
- If f and h were linear (∇f and ∇h_i constant), the updated λ_i would be exactly right. That is, in the next iteration we would exactly hit the constraint (by construction)
- The penalty term is like a measuring device for the necessary virtual gradient, which is generated by the augmentation term in the next iteration
- The λ_i are meaningful: they give the force/gradient that a constraint exerts on the solution

Algorithm 34 Augmented Lagrangian Equality method

Require: Input: initial $x\in\mathbb{R}^n$, function $f(x)$, $g(x)$, $\nabla f(x)$, $\nabla g(x)$, tolerances θ, ϵ
Require: Output: x
Require: Initialise $\mu=1$, $\lambda_i=0$
repeat
  find $x\leftarrow\arg\min_x f(x)+\mu\sum_{i=1}^{m}h_i^2(x)+\sum_{i=1}^{m}\lambda_i h_i(x)$
  $\forall i:\lambda_i\leftarrow\lambda_i+2\mu h_i(x^{'})$
until $|\Delta x|<\theta$ and $|h_i(x)|<\epsilon$

10.3.2.2 The inequality constraint

We then consider the inequality constraint in Equation (10.1.1), which we rewrite as

$$\min_x f(x)+\mu\sum_{i=1}^{m}I\big(g_i(x)\geqslant 0\vee\lambda_1(0)\big)g_i^2(x)+\sum_{i=1}^{m}\lambda_i g_i(x)$$

Note the λ_i are zero or positive but never negative. A constraint is either active or inactive:

- When active $\big(g_i(x)\geqslant 0\vee\lambda_1(0)\big)$ we aim for equality $g_i(x)=0$
- When inactive $\big(g_i(x)\geqslant 0\wedge\lambda_1(0)\big)$ we do not penalise / augment equality $g_i(x)=0$

Detail is given in the Algorithm (35).

Algorithm 35 Augmented Lagrangian Inequality method

Require: Input: initial $x\in\mathbb{R}^n$, function $f(x)$, $g(x)$, $\nabla f(x)$, $\nabla g(x)$, tolerances θ, ϵ
Require: Output: x
Initialise $\mu=1$, $\lambda_i=0$
repeat
  find $x\leftarrow\arg\min_x f(x)+\mu\sum_{i=1}^{m}I\big(g_i(x)\geqslant 0\vee\lambda_1(0)\big)g_i^2(x)+\sum_{i=1}^{m}\lambda_i g_i(x)$
  $\forall i:\lambda_i\leftarrow\max\big(\lambda_i+2\mu h_i(x^{'}),0\big)$
until $|\Delta x|<\theta$ and $g_i(x)<\epsilon$

Electronic copy available at: https://ssrn.com/abstract=3307566

10.4 The Lagrangian approach

10.4.1 The KKT conditions

Given a constraint problem in Equation (10.1.1), we define the Lagrangian as

$$L(x, \lambda, \nu) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{l} \nu_j h_j(x) \tag{10.4.3}$$

where $\lambda_i \geqslant 0$, $\nu_j \in \mathbb{R}$ are called dual variables or Lagrange multipliers. We assume that f, g_i and h_j are convex and differentiable.
We will first focus on the case

$$L(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x)$$

The Lagrangian is useful when computing optima analytically. It implies the Karush-Kuhn-Tucker (KKT) conditions of optimality (see Kuhn et al. [1951]). That is, the KKT conditions encode the conditions for local minima described in Section (10.1.3). Note, the optima are necessarily at saddle points of the Lagrangian. The Lagrangian implies a dual problem, which is sometimes easier to solve than the primal one.
As discussed in Section (10.1.3), at the optimum there must be a balance between the cost gradient $-\nabla f(x)$ and the gradient of the active constraints $-\nabla g_i(x)$. Formally, for optimal $x : \nabla f(x) \in \text{Span}\{\nabla g_i(x)\}$ or for optimal x, there must exist λ_i such that

$$-\nabla f(x) = \sum_{i=1}^{m} (-\lambda_i \nabla g_i(x))$$

Theorem 10.4.1 *KKT conditions*
Given the above Lagrangian, for optimal x, the Karush-Kuhn-Tucker conditions must hold (necessary condition): $\exists \lambda$ such that

$$\begin{aligned} \nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x) &= 0 \textit{ force balance (or stationarity)} \\ \forall i : g_i(x) &\leqslant 0 \textit{ primal feasibility} \\ \forall i : \lambda_i &\geqslant 0 \textit{ dual feasibility} \\ \forall i : \lambda_i g_i(x) &= 0 \textit{ complementary slackness} \end{aligned}$$

plus positive definite constraints on $\nabla_{xx} L(x, \lambda)$

The first condition can be equivalently expressed as $\exists \lambda$ such that

$$\nabla_x L(x, \lambda) = 0$$

The second condition corresponds to

$$\nabla_{\lambda_i} L(x, \lambda) = g_i(x) \leqslant 0 \,, \forall i$$

The fourth condition corresponds to

$$\sum_{i=1}^{m} \lambda_i g_i(x) = 0$$

The KKT conditions imply

Electronic copy available at: https://ssrn.com/abstract=3307566

1. Case 1: inactive constraint
   - when $\lambda^* = 0$ then we have $L(x^*, \lambda^*) = f(x^*)$
   - condition KKT $[1] \Rightarrow \nabla_x f(x^*) = 0$
   - condition KKT $[2] \Rightarrow x^*$ is a feasible point
2. case 2: active constraint
   - when $\lambda^* > 0$ then we have $L(x^*, \lambda^*) = f(x^*) + \lambda^* g(x^*)$
   - condition KKT $[1] \Rightarrow \nabla_x f(x^*) = -\lambda^* \nabla_x g(x^*)$
   - condition KKT $[4] \Rightarrow g(x^*) = 0$
   - condition KKT $[4]$ also implies $\Rightarrow L(x^*, \lambda^*) = f(x^*)$

Thus, the Lagrangian can be viewed as the energy function that generates (for good choice of λ) the right balance between cost and constraint gradients. Note, this is like in the augmented Lagrangian approach, where however we have an additional augmented squared penalty that is used to tune the λ_i.
In the case of multiple equality and inequality constraints, the Karush-Kuhn-Tucker conditions must hold (necessary condition): $\exists \lambda$ such that

$$\begin{aligned} \nabla_x L(x, \lambda, \nu) &= 0 \text{ force balance (or stationarity)} \\ \forall i : g_i(x) &\leqslant 0 \text{ primal feasibility} \\ \forall i : \lambda_i &\geqslant 0 \text{ dual feasibility} \\ \forall i : \lambda_i g_i(x) &= 0 \text{ complementary slackness} \\ h(x) &= 0 \text{ primal feasibility} \\ \text{plus positive definite constraints on } \nabla_{xx} L(x, \lambda) & \end{aligned}$$

***Remark* 10.4.1** *Concerning the stationarity condition: for a differentiable function f, we can not use $\partial f(x) = \{\nabla f(x)\}$ unless f is convex.*

10.4.2 Implication from the KKT conditions

10.4.2.1 Equality constraints

In the case of the equality, the Lagrangian is given by

$$L(x, \nu) = f(x) + \sum_{j=1}^{l} \nu_j h_j(x) = f(x) + \nu^\top h(x)$$

Given the solution in Section (10.1.3), we get the following relations

- $\min_x L(x, \nu) \Rightarrow 0 = \nabla_x L(x, \nu) \longleftrightarrow$ force balance
- $\max_\nu L(x, \nu) \Rightarrow 0 = \nabla_\nu L(x, \nu) = h_i(x) \longleftrightarrow$ constraint

As a result, the optima (x^*, ν^*) are saddle points where

- $\nabla_x L = 0$ ensures force balance, and
- $\nabla_\nu L = 0$ ensures the constraint

Electronic copy available at: https://ssrn.com/abstract=3307566

10.4.2.2 Inequality constraints

In the case of the inequality, the Lagrangian is given by

$$L(x,\lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) = f(x) + \lambda^\top g(x)$$

and we get

$$\max_{\lambda \geqslant 0} L(x,\lambda) = \begin{cases} f(x) \text{ if } g(x) \leqslant 0 \\ \infty \text{ otherwise} \end{cases}$$

and

$$\max_{\lambda_i \geqslant 0} L(x,\lambda) = \begin{cases} \lambda_i = 0 \text{ if } g(x) < 0 \\ 0 = \nabla_{\lambda_i} L(x,\lambda) = g_i(0) \text{ otherwise} \end{cases}$$

It implies either $\big(\lambda_i = 0 \wedge g_i(x) < 0\big)$ or $g_i(0) = 0$, which is exactly equivalent to the KKT conditions. Again, the optima (x^*, λ^*) are saddle points where

- $\min_x L$ enforces force balance, and
- $\max_\lambda L$ enforces the KKT conditions

10.4.3 The Lagrangian dual problem

10.4.3.1 Definition

We define the Lagrange dual function as

$$l(\lambda,\nu) = \inf_x L(x,\lambda,\nu)$$

which implies two problems:

1. primal problem: $\min_x f(x)$ such that $g(x) \leqslant 0$ and $h(x) = 0$
2. dual problem: $\max_{\lambda,\nu} l(\lambda,\nu)$ such that $\lambda \geqslant 0$

Note, the dual problem is convex, even if the primal problem is non-convex. That is, $l(\cdot,\cdot)$ is concave. Thus, we can rewrite the two problems as

1. primal problem: $\min_x \big[\sup_{\lambda \geqslant 0,\nu} L(x,\lambda,\nu)\big]$
2. dual problem: $\max_{\lambda \geqslant 0,\nu} \big[\inf_x L(x,\lambda,\nu)\big]$

The primal problem is equivalent to a min-max optimisation because $\max_{\lambda \geqslant 0,\nu} L(x,\lambda,\nu)$ ensures the constraints. For example, consider a two-player game. If player 1 chooses x that violates a constraint $g_1(x) > 0$, player 2 choose $\lambda_1 \to \infty$ so that $L(x,\lambda,\nu) = \cdots + \lambda_1 f_1(x) + \cdots \to \infty$. Therefore, player 1 is forced to satisfy the constraints.

Electronic copy available at: https://ssrn.com/abstract=3307566

10.4.3.2 Relation between primal and dual solutions

We now discuss the relation between primal and dual solutions in the case of inequality constraints. For any $\lambda_i \geqslant 0$, the dual function is always a lower bound

$$l(\lambda) = \min_x L(x,\lambda) \leqslant \min_x f(x) \text{ such that } g(x) \leqslant 0$$

and consequently

$$\max_{\lambda \geqslant 0}\big[\min_x L(x,\lambda)\big] \leqslant \min_x\big[\max_{\lambda \geqslant 0} L(x,\lambda)\big]$$

Including equality and inequality, we get:

Lemma 10.4.1 *Weak duality*
If $\lambda \geqslant 0$, then

$$l(\lambda,\nu) \leqslant f(x^*)$$

Proof: We have

$$\begin{aligned} l(\lambda,\nu) &= \inf_x L(x,\lambda,\nu) \leqslant L(x^*,\lambda,\nu) \\ &= f(x^*) + \sum_{i=1}^m \lambda_i g_i(x^*) + \sum_{j=1}^l \nu_j h_j(x^*) \leqslant f(x^*) \end{aligned}$$

Strong duality holds if and only if

$$\max_{\lambda \geqslant 0,\nu}\big[\min_x L(x,\lambda,\nu)\big] = \min_x\big[\max_{\lambda \geqslant 0,\nu} L(x,\lambda,\nu)\big]$$

If the primal is convex, and there exists an interior point, we get the Slater condition

$$\exists x : \forall i : g_i(x) < 0 \text{ and } \forall j : h_j(x) = 0$$

then we have strong duality. That is, primal and dual solutions are equivalent.

Theorem 10.4.2 *Strong duality*
For reasonable convex problems,

$$\sup_{\lambda \geqslant 0,\nu} l(\lambda,\nu) = f(x^*)$$

10.4.3.3 Interpretation

We can interpret duality as a linear approximation. We let $I_-(a) = \infty$ if $a > 0$ and 0 otherwise. We let $I_0(a) = \infty$ unless $a = 0$. We can rewrite the problem as

$$\min_x f(x) + \sum_{i=1}^m I_-(g_i(x)) + \sum_{j=1}^l I_0(h_j(x))$$

We replace $I_-(g_i(x))$ with $\lambda_i g_i(x)$, a measure of displeasure when $\lambda_i \geqslant 0$, $g_i(x) > 0$. Further, $\nu_j h_j(x)$ is a lower bound for $I_0(h_j(x))$. Thus, we get

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\min_x f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{l} \nu_j h_j(x)$$

As an example, we consider the linearly constrained least squares. The problem is as follows:

$$\min_x \frac{1}{2}\|Ax - b\|^2 \text{ such that } Bx = d$$

From the Lagrangian, we get

$$L(x, \nu) = \frac{1}{2}\|Ax - b\|^2 + \nu^\top (Bx - d)$$

Take the infimum, we get

$$\nabla_x L(x, \nu) = A^\top Ax - A^\top b + B^\top \nu \Rightarrow x = (A^\top A)^{-1}\big(A^\top b - B^\top \nu\big)$$

Replace the value of x and we get a simple unconstrained quadratic problem

$$\inf_x L(x, \nu) = \frac{1}{2}\|A\big[(A^\top A)^{-1}\big(A^\top b - B^\top \nu\big)\big] - b\|^2 + \nu^\top \Big(B\big[(A^\top A)^{-1}\big(A^\top b - B^\top \nu\big)\big] - d\Big)$$

10.4.3.4 Summary

When we have a constrained optimisation problem which is hard to solve, we might consider the dual problem since it may have simpler constraints. Further the solution of the latter is also the solution of the former.
Given the primal feasible x and the dual feasible λ and ν, the quantity

$$f(x) - l(\lambda, \nu)$$

is called the duality gap between x and λ, ν. We get a certificate of optimality: if we have a feasible x and know the dual $l(\lambda, \nu)$, then

$$\begin{aligned} l(\lambda, \nu) \leqslant f(x^*) \leqslant f(x) \quad &\Rightarrow \quad f(x^*) - f(x) \geqslant l(\lambda, \nu) - f(x) \\ &\Rightarrow \quad f(x) - f(x^*) \leqslant f(x) - l(\lambda, \nu) \end{aligned}$$

Thus, if the duality gap is zero, then x is primal optimal (and λ, ν are dual optimal). That is, all these inequalities are actually equalities. It provides a stopping criterion for a numerical implementation: if $f(x) - l(\lambda, \nu) \leqslant \epsilon$, then we are guaranteed that $f(x) - f(x^*) \leqslant \epsilon$.
We now introduce necessity and sufficiency for an optimum solution:

- Necessity states that if x^* and λ^*, ν^* are primal and dual solutions, with zero duality gap, then x^*, λ^*, ν^* satisfy the KKT conditions.
- Sufficiency states that if there exists x^*, λ^*, ν^* that satisfy the KKT conditions, then

$$\begin{aligned} l(\lambda^*, \nu^*) &= f(x^*) + \sum_{i=1}^{m} \lambda_i^* g_i(*) + \sum_{j=1}^{l} \nu_j^* h_j(*) \\ &= f(x^*) \end{aligned}$$

  where the first equality holds from stationarity, and the second holds from complementary slackness. Therefore, duality gap is zero so that x^* and λ^*, ν^* are primal and dual optimal.

Electronic copy available at: https://ssrn.com/abstract=3307566

In summary, KKT conditions are

- always sufficient
- necessary under strong duality

One of the most important uses of duality is that, under strong duality, we can characterise primal solutions from dual solutions. That is, for a problem with strong duality (assume Slater's condition) x^* and λ^*, ν^* are primal and dual solutions $\Longleftrightarrow$ x^* and λ^*, ν^* satisfy the KKT conditions.

If $\min_x L(x, \lambda, \nu)$ can be solved analytically, so does the convex dual problem. More generally, we have

$$\text{Optimisation problem} \rightarrow \text{Solve KKT conditions}$$

Thus, we can apply standard algorithms for solving an equation system $r(x, \lambda) = 0$ such as the Newton method

$$\nabla r \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = -r$$

It leads to primal-dual algorithms that adapt x and λ concurrently. They use the curvature $\nabla^2 f$ to estimate the right λ to push out of the constraint.

10.4.3.5 Equivalence

In general, one switch back and forth between the constrained form

$$\min_x f(x) \text{ such that } g(x) \leqslant d \text{ (C)}$$

where $d \in \mathbb{R}$ is a tuning parameter, to the Lagrange form

$$\min_x f(x) + \lambda \cdot g(x) \text{ (L)}$$

where $\lambda \geqslant 0$ is a tuning parameter, and claim the two methods are equivalent. We get two cases

1. (C) to (L): if the problem (C) is strictly feasible, then strong duality holds, and there exists some $\lambda \geqslant 0$ (dual solution) such that any solution x^* in (C) minimises

   $$f(x) + \lambda \cdot \big(f(x) - d\big)$$

   so that x^* is also a solution in (L).
2. (L) to (C): if x^* is a solution in (L), then the KKT conditions for (C) are satisfied by taking $d = g(x^*)$ so that x^* is a solution in (C).

Thus, we get the equivalence

$$\begin{aligned} \bigcup_{\lambda \geqslant 0} \{ \text{ solutions in (L) } \} &\subseteq \bigcup_{d} \{ \text{ solutions in (C) } \} \\ \bigcup_{\lambda \geqslant 0} \{ \text{ solutions in (L) } \} &\supseteq \bigcup_{d \text{ such that (C) is strictly feasible}} \{ \text{ solutions in (C) } \} \end{aligned}$$

Note, this is not a perfect equivalence (albeit minor non-equivalence).
Using the KKT conditions and simple probability arguments, we get the following result:

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 10.4.3 *Let f be differentiable and strictly convex, $A \in \mathbb{R}^{n\times p}$, $\lambda > 0$. Consider*

$$\min_x f(Ax) + \lambda\|x\|_1$$

If the entries of A are drawn from a continuous probability distribution (on $\mathbb{R}^{n\times p}$), then with probability 1, the solution $x^* \in \mathbb{R}^p$ is unique and has at most $\min\{n, p\}$ nonzero components.

Note, while f must be strictly convex, there is no restrictions on the dimensions of A and one could choose $p >> n$.

10.4.4 Additional

10.4.4.1 The log barrier method revisited

Going back to the log barrier method in Section (10.3.1.1), for a given μ, the optimality condition is

$$\nabla f(x) - \sum_{i=1}^m \frac{\mu}{g_i(x)}\nabla g_i(x) = 0$$

or equivalently, we get the modified (approximated) KKT conditions

$$\nabla f(x) - \sum_{i=1}^m \lambda_i \nabla g_i(x) = 0 \ , \ \lambda_i g_i(x) = -\mu$$

We see that centring in the log barrier method is equivalent to solving the modified KKT conditions. In addition, on the central path, the duality gap is μ:

$$l(\lambda^*(\mu)) = f(x^*(\mu)) + \sum_i \lambda_i g_i(x^*(\mu)) = f(x^*(\mu)) - \mu$$

10.4.4.2 Non-convex optimisation

We consider the optimisation problem with equality constraint

$$\min_x f(x) \text{ such that } h(x) = 0$$

where both $f(\cdot)$ and $h(\cdot)$ are neither convex nor linear. Given the Lagrangian

$$L(x, \nu) = f(x) + \nu^\top h(x)$$

we get the min-max problem

$$\min_x \max_\nu L(x, \nu)$$

The idea is to let x_0 be an initial value of x and compute the increment dx and the value of ν by solving a linearised version of the above problem, replacing x by $x + dx$, and iterates. If the function $f(\cdot)$ is differentiable, the increment dx is normally computed at each iteration by solving the linear system

$$\begin{pmatrix} \eta I & \nabla_x h^\top \\ \nabla_x h & 0 \end{pmatrix} \begin{pmatrix} dx \\ \nu \end{pmatrix} = \begin{pmatrix} -\nabla_x f \\ -h(x) \end{pmatrix}$$

which amounts to projected gradient descent. The projection is performed onto the hyperplanes of linearised constraints. This system is derived from the KKT conditions that dx and ν must satisfy.

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, when the function $f(\cdot)$ is a sum of squared residuals (typical in regression problems), one can replace the projected gradient by Gauss-Newton (GN) or Levenberg-Marquardt (LM) (see Fua et al. [2010]). The iteration scheme remains the same but the KTT conditions become

$$\begin{pmatrix} J^\top J + \eta I & \nabla_x h^\top \\ \nabla_x h & 0 \end{pmatrix} \begin{pmatrix} dx \\ \nu \end{pmatrix} = \begin{pmatrix} -J^\top r(x_t) \\ -h(x_t) \end{pmatrix}$$

where r is the vector of residuals and $J = \nabla_x r$ is its Jacobian matrix evaluated at x_t.

10.5 Standard convex programs

There exist a large variety of mathematical programs, each being a different research area in itself with extensive theory and algorithms. Given the constraint problem in Equation (10.1.1), we review a few basic convex optimisation programming models and present some examples. See Nocedal et al. [1999] for details. See Press et al [1992] for numerical implementation.

10.5.1 Simple programming models

10.5.1.1 Quadratic programming

Quadratic programming is used in problems minimising least squares loss function. A quadratic program (QP) has a quadratic objective with linear constraints. The problem is stated as

$$\min_x \frac{1}{2} x^\top Q x + C^\top x \text{ such that } a_i x \leqslant b_i \text{ , } i \in I \text{ , } a_j x = b_j \text{ , } j \in \varepsilon$$

where the Hessian matrix Q is $n \times n$ symmetric, I and ε are finite sets of indices and a_i, $i \in I \bigcup \varepsilon$ are $n \times 1$ vectors. As discussed above, if the matrix Q is positive, $x^\top Q x \geqslant 0$ for any x, then the problem is convex. For convex QP, any local solution is also a global solution. A QP can always be solved or shown to be infeasible in a finite number of iterations. AS discussed in Section (10.4), the KKT optimal conditions are

1. $Qx + \sum_{i \in I} \lambda_i a_i + \sum_{j \in \varepsilon} \nu_j a_j = 0$ Dual Feasibility
2. $a_i^\top x \leqslant b_i$ for $i \in I$ Primal Feasibility
3. $a_j^\top x = b_j$ for $j \in \varepsilon$ Primal Feasibility
4. $\lambda_i \left(a_i^\top x - b_i \right) = 0$ for $i \in I$ Complementary

In the special case where there are no constraints ($I = 0$), then the KKT point can be found by simply solving a system of linear equations. It is more difficult in presence of inequality constraints. Two methods exists

1. interior point methods
2. active-set methods: the optimal active set is the set of constraints satisfied as equalities at the optimal solutions. Active set methods work by making educated guesses as to the active set and solving the resulting equality constrained QP. In the case of wrong guess, one can use gradient and Lagrangian multiplier information to determine constraints to add to or subtract from the active set.

Electronic copy available at: https://ssrn.com/abstract=3307566

10.5.1.2 Linear programming

Linear programming optimises a linear function subject to linear constraints. Since linear functions and constraints are convex, a linear program (LP) is always a convex program. Linear programming is a special case of the QP above, with the Hessian Q equal to zero. The problem is stated as

$$\min_x C^\top x \text{ such that } a_i x \leqslant b_i \text{ , } i \in I \text{ , } a_j x = b_j \text{ , } j \in \varepsilon$$

In general, one uses interior point methods and simplex methods (active set methods) to solve LP problems.

10.5.1.3 Second-order cone programming

The second-order cone programming (SOCP) problems have a linear objective, second-order cone constraints, and possibly additional linear constraints. The problem is stated as

$$\min_x C^\top x \text{ such that } \|R_i x + d_i\|_2 \leqslant a_i x + b_i \text{ , } i \in I \text{ , } a_j x = b_j \text{ , } j \in \varepsilon$$

where $R_i \in \mathbb{R}^{n_i \times n}$ and $d_i \in \mathbb{R}^{n_i}$. These problems are often solved by using interior point algorithms.

10.5.1.4 Semidefinite programming

Semidefinite programs (SDP) are the generalisation of linear programs (LP) to matrices. In standard form, a SDP minimises a linear function of a matrix subject to linear equality constraints and a matrix non-negativity constraint. The problem is stated as

$$\min_X < C, X > \text{ such that } < A_i, X >= b_i \text{ , } i \in I \text{ , } X \succeq 0$$

where X, C and A_i take values in $\mathbb{R}^{n \times n}$ and $b_i \in \mathbb{R}$. Further, $X \succeq 0$ means X must be positive semidefinite and $< C, X >= \text{trace}(CX)$. In general, SDP are solved via interior programming methods (see Mittelmann [2003]).

10.5.2 Some examples

10.5.2.1 Quadratic programming

As an example we consider a quadratic function with equality constraints. Given $Q \geqslant 0$, the optimisation problem is

$$\min_x \frac{1}{2} x^\top Q x + c^\top x \text{ such that } Ax = 0$$

It is a convex problem with no inequality constraints so that by the KKT conditions x is a solution if and only if

$$\begin{pmatrix} Q & A^\top \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} -c \\ 0 \end{pmatrix}$$

for some λ. Linear system combines stationarity and primal feasibility (complementary slackness and dual feasibility are vacuous).

Electronic copy available at: https://ssrn.com/abstract=3307566

10.5.2.2 The Lasso problem

Another example is the Lasso problem: given the response $y \in \mathbb{R}^n$, the predictors $A \in \mathbb{R}^{n \times p}$ (columns $A_1, ..., A_p$), the optimisation problem is

$$\min_{x \in \mathbb{R}^p} \frac{1}{2} \|y - Ax\|^2 + \nu \|x\|_1$$

The KKT conditions are

$$A^\top (y - Ax) = \lambda s$$

where $s \in \partial \|x\|_1$, that is,

$$s_i \in \begin{cases} \{1\} \text{ if } x_i > 0 \\ \{-1\} \text{ if } x_i < 0 \\ (-1, 1] \text{ if } x_i = 0 \end{cases}$$

Note, if $|A_i^\top (y - Ax)| < \nu$ then $x_i = 0$.

10.6 Generalising the penalty function approach

10.6.1 The energy function

As already seen above, the penalty function method consists of transforming the constrained optimisation problem in Equation (10.1.1) (but with $g_i(x) \geqslant 0$) into an unconstrained one based on a penalty function (see Luenberger [1984]). That is, the penalty function $E(\cdot, k)$ satisfies

$$E(x, k) = \alpha f(x) + \sum_{i=1}^{m} k_i P(g_i(x))$$

where $\alpha = +1$ for minimisation problems and $\alpha = -1$ for maximisation problems. The penalty term $P(g_i(x))$ should satisfy

$$\begin{aligned} P(g_i(x)) &= 0 \text{ if } g_i(x) \geqslant 0 \text{ the corresponding constraint is satisfied} \\ P(g_i(x)) &> 0 \text{ if } g_i(x) < 0 \end{aligned}$$

The constants k_i, $i = 1, ..., m$, define the relative weight concerning the satisfaction of some constraints against some other and/or the relative weight of satisfying all the constraints or minimising $sf(x)$. The task of tuning these parameters provides a means of customising the problem according to the current needs.
As discussed in Section (4.1.1), the function $E(\cdot, k)$ is also called the energy function for the corresponding network (see Cichocki et al. [1993]).
Assuming differentiability, a local minimum of the penalty function is obtained by using the dynamic gradient scheme

$$\frac{dx}{dt} = -\mu \nabla_x E(x, k) \, , x(0) = x^{(0)}$$

where t is the time step and

$$\mu = \text{diag}(\mu_1, \mu_2, ..., \mu_n)$$

or in extended form

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}
\frac{dx_1}{dt} &= -\mu_1\Big(\frac{\partial f(x)}{\partial x_1} + \sum_{i=1}^{m} k_i \frac{\partial P}{\partial g_i}\frac{\partial g_i(x)}{\partial x_1}\Big), x_1(0) = x_1^{(0)} \\
\frac{dx_2}{dt} &= -\mu_2\Big(\frac{\partial f(x)}{\partial x_2} + \sum_{i=1}^{m} k_i \frac{\partial P}{\partial g_i}\frac{\partial g_i(x)}{\partial x_2}\Big), x_2(0) = x_2^{(0)} \\
&\vdots \\
\frac{dx_n}{dt} &= -\mu_n\Big(\frac{\partial f(x)}{\partial x_n} + \sum_{i=1}^{m} k_i \frac{\partial P}{\partial g_i}\frac{\partial g_i(x)}{\partial x_n}\Big), x_n(0) = x_n^{(0)}
\end{aligned}$$

where $\mu_j > 0$ and $k_i > 0$. Usually one takes $\mu_j = \mu = \frac{1}{\tau}$, $j = 1, ..., m$, where τ is a time constant, and $k_i = k$, $i = 1, ..., n$.

10.6.2 The penalty functions

The penalty function has the general form

$$P(g(x)) = \Psi(g(x))$$

where the function $\Psi(x)$ should penalise only configurations where $x > 0$. One possibility is to use a sigmoid function or

$$\Psi(x) = x\Theta(x)$$

where $\Theta(x)$ is a Heaviside function (see Ohlsson [1993]). The penalty term is zero if and only if the constraint is satisfied, otherwise the penalty is proportional (linear) to the degree of violation. The slope of $\Psi(x)$ is implicitly given by the strength of the constraint k in the energy function.
Some examples of penalty terms are

$$\begin{aligned}
P(g_i(x)) &= \big[\min(0, g_i(x))\big]^2 \\
P(g_i(x)) &= -\min(0, g_i(x))
\end{aligned}$$

10.6.2.1 Type I

In the first case, the energy function becomes

$$E(x, k) = f(x) + \frac{1}{2}\sum_{i=1}^{m} k_i\big[\min(0, g_i(x))\big]^2$$

Assuming that $g_i(x)$, $i = 1, ..., m$, have continuous first derivatives, one can show that the same is true for $P(g_i(x))$, so that the energy function is continuously differentiable. The gradient of $P(g_i(x))$ is

$$\nabla_x\big[\min(0, g_i(x))\big]^2 = \min(0, g_i(x))\nabla_x g_i(x)$$

Using the gradient strategy for minimising the energy function $E(\cdot, k)$, we get the system of ordinary differential equations

$$\frac{dx_j}{dt} = -\mu_j\Big(\frac{\partial f(x)}{\partial x_j} + \sum_{i=1}^{m} k_i S_i g_i(x)\frac{\partial g_i(x)}{\partial x_j}\Big), j = 1, ..., n$$

Electronic copy available at: https://ssrn.com/abstract=3307566

where the control signals are

$$S_i = \begin{cases} 1 \text{ if } g_i(x) \leqslant 0 \\ 0 \text{ if } g_i(x) > 0 \end{cases}$$

and

$$S_i g_i(x) = \min(0, g_i(x))$$

Kennedy et al. [1988] and Cichocki et al. [1993] showed that the functional scheme for simulating these equations could be considered as a neural network (NN), where the integrators represent the neurons (basic units) and functional nonlinear generators build up the connections between them. This approach replaces the constrained problem in Equation (10.1.1) by an unconstrained minimisation of the differentiable penalty function $E(\cdot, k)$. Luenberger [1984] explained that theoretical results on the penalty function method show that equivalence of the problems in Equation (10.1.1) and $\min_x E(x, k)$ is only obtained in the limit, as

$$\min_{i=1,...,m} \{k_i\} \to \infty$$

Since the values of the parameters k_i, $i = 1, ..., m$, used in a NN are finite, it follows that the unconstrained minima of the energy function obtained by the NN will only be approximations to the true solutions of the constrained problem.

10.6.2.2 Type II

We now consider the case of the second penalty term, the energy function becomes

$$E(x, k) = f(x) - \sum_{i=1}^{m} k_i \min(0, g_i(x))$$

Due to the non-differentiability of the penalty term, this penalty function is non-differentiable, even though the functions $g_i(x)$ are assumed differentiable. Luenberger [1984] proved that any unconstrained minimum of the above penalty function is also a solution of the constrained problem in Equation (10.1.1) provided that $\min_{i=1,...,m}\{k_i\}$ is sufficiently large. Thus, an unconstrained minimisation will yield the true solution if k_i, $i = 1, ..., m$ are sufficiently large but finite. This solution is termed exact. In that case, the corresponding system of ordinary differential equations is

$$\frac{dx_j}{dt} = -\mu_j \Big(\frac{\partial f(x)}{\partial x_j} - \sum_{i=1}^{m} k_i S_i \frac{\partial g_i(x)}{\partial x_j} \Big), j = 1, ..., n$$

where $\mu_j > 0$ and the control signals are

$$S_i = \begin{cases} 1 \text{ if } g_i(x) \leqslant 0 \\ 0 \text{ if } g_i(x) > 0 \end{cases}$$

The functional scheme is a simpler NN than with the previous penalty function using simple switches instead of the expensive analog multipliers. However due the non-smooth nature of the penalty function, the first derivative discontinuities can lead to parasitic effects slowing up the solution speed.

10.6.2.3 Type III

Rather than adding together the penalty terms $-k_i \min(0, g_i(x))$, Mladenov et al. [1999] proposed to take the maximum of these terms, getting the penalty (energy) function

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} E(x,k) &= f(x) + \max_{i=1,...,m}\big(k_i P(g_i(x))\big) \\ &= f(x) + \max_{i=1,...,m}\big(\max\big(0, -k_i g_i(x)\big)\big) \\ &= f(x) + \max\big(0, -k_1 g_1(x), -k_2 g_2(x), ..., -k_m g_1(m)\big) \end{aligned}$$

In that setting, only the most violated inequality constraint is taken into account. Note, this system is non-differentiable. Again, it has been shown that it is an exact penalty function (see Polak [1997]).
To apply the gradient strategy, we need to select

$$\frac{\partial x}{\partial t} \in -\mu \partial E(x,k)$$

where $\partial E(x,k)$ is the generalised gradient of the penalty function $E(\cdot,k)$. We use the same system of ordinary differential equations as in Type II except that the control signals S_i are defined as follows:

$$S_i = \begin{cases} 1 \text{ if } \max\big(0,-g_i(x)\big) = \max_{j=1,...,m}\big(\max\big(0,-g_i(x)\big)\big) \\ 0 \text{ if } \max\big(0,-g_i(x)\big) < \max_{j=1,...,m}\big(\max\big(0,-g_i(x)\big)\big) \end{cases}$$

The proposed NN architecture consists in a winner-take-all block included into the Control Network (see Cichocki et al. [1993]). This block is used to determine dynamically the index i^* of the most violated inequality and to generate the control signal $S_{i*} = 1$ corresponding to this inequality, while all other control signals (corresponding to other inequalities) are equal to 0. The same non-smooth character of the exact penalty function as in Type II applies.

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 11

Global search optimisation

11.1 Evolutionary algorithms

11.1.1 A brief history of evolutionary algorithms

Evolutionary algorithms (EAs) introduced by Holland [1962] [1975] and Fogel [1966] are robust and efficient optimisation algorithms based on the theory of evolution proposed by Darwin [1882], where a biological population evolves over generations to adapt to an environment by mutation, recombination and selection. They are stochastic search algorithms, searching from multiple points in space instead of moving from a single point like gradient-based methods do. These algorithms are typically initiated with a population of potential solutions, that may be drawn randomly or specified prior to the beginning of the search. Iteration on the population is based on the principles of natural selection, with each iteration, or generation, improving in fitness as defined by some pre-determined measure. The methods by which each generation is determined are specific to the particular algorithm in question, however, all evolutionary algorithms rely on classes of stochastic operator known as selection and reproduction operators. The selection operator acts to ensure that individuals with greater fitness in each generation are selected as parents for the next generation, whilst the reproduction operator determines how the next generation is derived from the parents selected. Moreover, they work on function evaluation alone (fitness) and do not require derivatives or gradients of the objective functions. McKay [2008] claimed that despite their simplicity, given modest resources and a relatively tough optimisation problem, Evolutionary Algorithms reliably converge to good solutions, and what's more, they are suited to parallel implementation due to their speed scaling almost linearly with the number of processors.

There is a large literature describing different evolutionary algorithms (EAs) commonly used to solve constrained non-linear programming problems (CNOPs) such as evolutionary programming, evolution strategies, genetic algorithms (GAs), differential evolution (DE) and many more. For instance, GAs are general purpose search algorithms based on an evolutionary paradigm where the population members are represented by strings, corresponding to chromosomes. Search starts with a population of randomly selected strings, and, from these, the next generation is created by using genetic operators (mutation). At each iteration individual strings are evaluated with respect to a performance criteria and assigned a fitness value. Strings are randomly selected using these fitness values to either survive or to mate to produce children for the next generation. However, among the different EA's commonly used DE became very popular. DE is a population-based approach to function optimisation generating a new position for an individual by calculating vector differences between other randomly selected members of the population. The DE algorithm is found to be a powerful evolutionary algorithm for global optimisation in many real problems. As a result, since the original article of Storn and Price [1995] many authors improved the DE model to increase the exploration and exploitation capabilities of the DE algorithm when solving optimisation problems.

Since EAs are search engines working in unconstrained search spaces they lacked until recently of a mechanism to

Electronic copy available at: https://ssrn.com/abstract=3307566

deal with the constraints of the problems. The first attempts to handle the constraints were either to incorporate methods from mathematical programming algorithms within EAs such as penalty functions, or, to exploit the mathematical structure of the constraints. Then, a considerable amount of research proposed alternative methods to improve the search of the feasible global optimum solution. Most of the research on DE focused on solving CNOPs by using a sole DE variant, a combination of variants or combining DE with another search method. One of the most popular constraint handling mechanisms is the use of the three feasibility rules proposed by Deb [2000] on genetic algorithms. Using some of the improvements to the DE algorithm combined with simple and robust constraint handling mechanisms we propose a modified algorithm for solving our optimisation problem under constraints which greatly improves its performances.

11.1.2 Some optimisation methods

Before detailing differential evolution (DE), we introduce a few alternative optimisation methods (see Witkowski [2011] and Press et al. [1992] for numerical implementation). For simplicity of exposition we define a few operators used in several optimisation methods. We let $R_U(a,b)$ to return a random uniformly distributed value between a (inclusive) and b (exclusive), and $R_N(\mu,\sigma)$ to return a random normally distributed value drawn from a normal distribution with mean μ and standard deviation σ. Further, we define the mutation operator $M(X,\delta)$, where X is a vector of size N and δ is the mutation strength, as follow

```
Begin
 i ← R_U(1, N)
 X_i ← ± R_U(0, 1). δ
return X
End
```

11.1.2.1 Random optimisation

Random optimisation is an iterative optimisation method designed for single objective problem which consists in assigning uniformally distributed random values to an individual. In each iteration, the individual is modified by adding a normally distributed vector to it, in the case where the resulting individual is better than the source individual. We let $f : \mathbb{R}^N \to \mathbb{R}$ be the fitness function subject to optimisation, and we let $X \in \mathbb{R}^N$ be a position in the search space. The vector X is initialised as follow

$$X \leftarrow [R_{U,1}(0,1), ..., R_{U,N}(0,1)]$$

and the algorithm satisfies

```
Begin
while not terminationCriterion() do
    X^' ← X + [R_N, 1(μ, σ), ..., R_N, N(μ, σ)]
   if f(X^') > f(X) then
       X ← X^'
   end if
end while
End
```

When the value of an individual is subject to a constraint, we consider four strategies where one or more genes do not fit the constraints.

- Dropping the individual altogether: the algorithm leaves the old value of the individual.
- Dropping only the offending gene: the old gene is left.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Trimming the gene to the constraint: trimming either to the left or to theright value of the constraint.
- Bounce back: if g_{min} and g_{max} are the minimum and maximum values of the gene, respectively, then do

```
Begin
while not notConstraints(g) do
   if g > g_max then
       g ← g_max − |g − g_max|
   else
       g ← g_min + |g_min − g|
   end if
end while
End
```

11.1.2.2 Particle swarm optimisation

Particle swarm optimisation is an optimisation algorithm simulating swarming/social behaviour of agents (particles). In this scheme, a single particle is a solution candidate flying through the search space with some velocity, remembering its best position, and learning the best position known to its neighbour particles. While traveling the search space, the particles adjust their speed (both direction and value) based on their personal experiences and on the knowledge of their neighbourhood particles. Since various schemes for determining particle neighbourhood can be imagined, we can discern

- the global neighbourhood: all particles are neighbours to each other.
- the neighbourhood determined by Euclidean distance.
- the neighbourhood determined by normalised Euclidean distance.

where the normalisation process adjust the size of the neighbourhood so that the dimension of the search space is accounted for. It is a parallel direct search method using NP parameter vector

$$X_i \text{ for } i = 1, .., NP$$

as a population with vector of velocity V. The initialisation is given by

$$X_0...X_{NP} \leftarrow [R_{U,1}(0,1), ..., R_{U,N}(0,1)]$$
$$V_0...V_{NP} \leftarrow [R_{U,1}(0,1), ..., R_{U,N}(0,1)]$$

and the algorithm is as follow

```
Begin
while not terminationCriterion() do
   for i = 1 to NP do
      for j = 1 to N do
          φ_1 ← R_U(0,1)
          φ_2 ← R_U(0,1)
          V_ij^' ← w V_ij + σ ( φ_1 C_1 (best(p_i)_j − X_ij)
          + φ_2 C_2 (best_Nhood(p_i)_j − X_ij) )
      end for
```


Electronic copy available at: https://ssrn.com/abstract=3307566

```
      if  V^' > V_max  then
          V^' ← V_max
      end if
      if  V^' < −V_max  then
          V^' ← −V_max
      end if
       X_i^' ← X_i + V_i^'
   end for
 X ← X^'
 V ← v^'
end while
End
```

where N is the number of dimension of the search space, X is a vector of particle positions, V is a vector of particle velocities (each velocity is a vector with N elements), V_{max} is the maximum possible velocity, w is the velocity-weight or inertia factor (how much the particle will base its next velocity on its previous one), σ is the position weight determining the importance of the particles position to the next particle velocity, ϕ_i for $i = 1, 2$ are uniform random variables taken as an additional weight when determining the next particle velocity, C_1 is the self confidence weight (or self learning rate), C_2 is the swarm confidence weight (or neighbourhood learning rate), $best_{Nhood}(p)$ is an operator returning the best (most fit) position known to the particle and its neighbourhood, and $best(p)$ is an operator returning the most fit position that a particle has visited.

11.1.2.3 Cross entropy optimisation

The cross entropy optimisation consists of two steps

1. generating a sample population from a distribution.
2. updating the parameters of the random mechanism to produce a better (fitter) sample in the next population.

This behaviour conceptually substitutes the problem of finding an optimal individual to that of iteratively finding a random distribution that generate good individuals. We start by using a random Gaussian distribution to produce variables, and then improve the distribution by means of importance sampling, finding a new distribution from the best samples. We let N be the number of dimensions of the search space, N_{size} be the sample size, $(X_1, .., X_N)$ denotes the optimisation population, and we assume that the population is sorted so that

$$\forall i = 1, .., N-1 \ , f(X_i) \geqslant f(X_{i+1})$$

We then define the mean and standard deviation as

$$\begin{aligned} mean(X, N_{sample}) &= \frac{1}{N_{sample}} \sum_{i=0}^{N_{sample}} X_i \\ Std(X, N_{sample}) &= \sqrt{\frac{1}{N_{sample}} \sum_{i=0}^{N_{sample}} (X_i - mean(X, N_{sample}))^2} \end{aligned}$$

Since X is a vector, then $mean(X)$ and $Std(X)$ work on a set of vectors and return a vector of elements, that is, a vector of means and standard deviations of columns of the given input matrix. Additional importance sampling with the size of N_{sample} is performed on the given set. The initialisation is as follow

$$X_0 ... X_{NP} \leftarrow [R_{U,1}(0,1), ..., R_{U,N}(0,1)]$$

Electronic copy available at: https://ssrn.com/abstract=3307566

and

$$\mu \leftarrow mean(X, N_{sample})$$
$$\sigma \leftarrow Std(X, N_{sample})$$

The algorithm is as follow

```
Begin
while not terminationCriterion() do
   for i = 1 to NP do
       X_i ← [R_N, 1(μ_1, σ_1), ..., R_N, N(μ_N, σ_N)]
   end for
 μ ← mean(X, N_sample)
 σ ← Std(X, N_sample)
end while
End
```

In addition, the values of the mean μ and standard deviation σ can be smoothed over time (from iteration to iteration), improving the convergence of the algorithm.

11.1.2.4 Simulated annealing

Simulated annealing is a probabilistic optimisation method inspired by the physical process of annealing metals where a metal is slowly cooled so that its structure is frozen in a minimal energy configuration (see Tsitsiklis et al. [1993]). The algorithm is similar to hill climbing where an individual will iteratively go to a better neighbourhood position (picked at random), additionally an individual may go to a worse position with a probability proportional to its temperature. The probability that an individual will go to a worse position is calculated by the Boltzmann probability factor $e^{-\frac{E(pos)}{\frac{T}{k_B}}}$ where $E(pos)$ is the energy at the new position (calculated as a difference of fitness of two positions), k_B is the Botzmann constant $(1.38065052410^{-23}J/K)$, and T is the temperature of the solid. The rate at which the solid is frozen is called a cooling schedule, and we consider two variants

1. $$getTemp(t) = T_{start}(a^t)$$
   where T_{start} is the starting temperature, t is the current iteration, and a is a parameter of the cooling schedule. Further, $a > 0 \wedge a \approx 0$.

2. $$getTemp(t) = T_{start}(1 - \epsilon)^{\frac{t}{m}}$$
   where ϵ and m are parameters of the cooling schedule. Further, $0 < \epsilon \leqslant 1$.

The initialisation is as follow:

$$X \leftarrow [R_{U,1}(0, 1), .., R_{U,N}(0, 1)]$$

and

$$X_{best} \leftarrow X$$

We let $f : \mathbb{R}^N \rightarrow \mathbb{R}$, and $M(X, \delta)$ be the mutation operator and define the algorithm as follow:

Electronic copy available at: https://ssrn.com/abstract=3307566

```
Begin
while not  terminationCriterion()  do
    X^' ← M(X, δ)
    Δ E ← f(X) − f(X^')
      if  Δ E ⩽ 0  then
          X ← X^'
         if  (f(X)) > f(X_best)  then
             X_best ← X
         end if
      else
          T ← getTemp(t)
         if  R_U(0, 1) < e^ Δ E / k_B^t  then
             X ← X^'
         end if
      end if
 t ← t + 1
End
```

11.1.2.5 Introduction to genetic algorithms

GA is an optimization technique that produces optimisation of the problem by using natural evolution based on survival of the fittest. The search space is initialised with a set of solution called chromosome, which is a set of genes. Quality and fitness of a chromosome is measured by fitness function. The behaviour of Genetic algorithm is determined by exploration and exploitation with the help of operators like reproduction (selection), crossover (recombination) and mutation.
The general procedure for genetic algorithm is (see Witten et al. [2005]):

1. START: Generate random population.
2. FITNESS: Evaluate the fitness f(x) of each chromosome x in the population.
3. NEW POPULATION: Create a new population by repeating following steps until the new population is complete
   - REPRODUCTION OR SELECTION: Parents chromosomes are selected from population according to their fitness to crossover and produce new offspring.
   - CROSSOVER: crossover operator produce new two offspring from selected two parents based crossover probability.
   - MUTATION: Mutation operator produce new offspring by mutate single bit position in chromosome. Mutation used to maintain genetic diversity.
4. ACCEPTING: Place new offspring in the new population.
5. REPLACE: Use new generated population for a further run of the algorithm
6. TEST: If the end condition is satisfied, stop, and return the best solution in current population.
7. LOOP: Go to stop.

Performance of GA depends on various parameters like population size, crossover and mutation rate and computation time. Performance of genetic algorithm can be optimized by parallel genetic algorithm. Parallel Genetic Algorithm (PGA) is an algorithm that works by dividing a large problem into smaller tasks. Genetic algorithm with parallel processing reduces the computation time.
Some of the advantages and disadvantages of GA are:

Electronic copy available at: https://ssrn.com/abstract=3307566

1. Advantages of GA
    - GA has faster and more efficient as compared to the traditional methods.
    - It has very good parallel capabilities.
    - It optimises both continuous and discrete functions and also multi-objective problem.
    - It is Useful when the search space is very large and there are a large number of parameters involved.
2. Disadvantage of GA
    - Fitness values are calculated repeatedly which might be computationally expensive for some problems.
    - Being stochastic, there is no guarantee of the optimality or the quality of the solution.

When using GAs with classification algorithms, some of the issues and challenges are

- Local Convergence: GA sometimes converge on local optima, incorrect peak populate due to sampling errors. One needs to modify selection pressure to prevent premature convergence.
- Parameter Setting: GA parameters such as selection rate, chromosome length, population size, crossover probability, mutation probability and total number of generations has a large impact on performance of genetic algorithm. Setting these parameters is a complex task.

11.1.3 Introduction to differential evolution

11.1.3.1 The DE algorithm

According to Feoktistov [2006], Differential Evolution (DE) is first and foremost an optimisation algorithm, and in particular, one of the most powerful tools for global optimisation, regardless of it simplicity. It is so named because it identifies differences in individuals through the use of a simple and fast linear operator (differentiation), and in doing so, realises the evolution of a population of individuals in some intelligent manner. That is, the main characteristic of DE is an adaptive scaling of step sizes resulting in fast convergence behaviour. The Differential Evolution (DE) proposed by Storn and Price [1995] is an algorithm that can find approximate solutions to nonlinear programming problems. It is a parallel direct search method using NP parameter vectors, where each vector (or individual) is of dimension D equals the number of objective function parameters. The vectors

$$X_{i,G} \text{ for } i = 0, .., NP - 1$$

forms a population for each generation G, like any evolutionary algorithms. The initial vector population is chosen randomly, covering the entire parameter space. The number of vectors NP is a function of the dimension D, such as $NP \in [5D, 10D]$ but NP must be at least 4 to ensure that DE will have enough mutually different vectors to play with (see Storn et al. [1997]). Each of the NP parameter vectors undergoes mutation, recombination and selection.

11.1.3.1.1 The mutation The role of mutation is to explore the parameter space by expanding the search space giving its name to the DE algorithm. For a given parameter vector $X_{i,G}$ called the Target vector, the DE generates a Donor vector V made of three or more independent parent vectors $X_{r_l,G}$ for $l = 1, 2, ..$ where r_l is an integer chosen randomly from the interval $[0, NP - 1]$ and different from the running index i. In the spirit of Wright [1991], the main idea is to perturbate a Base vector $\hat{V}$ with a weighted difference vector (called differential vectors)

$$V = \hat{V} + F \sum_{l=1} \left(X_{r_{2l-1},G} - X_{r_{2l},G} \right) \tag{11.1.1}$$

where the mutation factor F is a constant taking values in $[0, 2]$ and scaling the influence of the set of pairs of solutions selected to calculate the mutation value. Most of the time, the Base vector is defined as the arithmetical crossover operator

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\hat{V} = \lambda X_{best,G} + (1 - \lambda) X_{r_1,G}$$

where $\lambda \in [0, 1]$ allows for a linear combination between the best element $X_{best,G}$ of the parent population vectors and a randomly selected vector $X_{r_1,G}$. It is called a global selection when $\lambda = 1$ while when $\lambda = 0$ the base vector is the same as the target vector, $X_{r_1,G} = X_{i,G}$ and we get a local selection. In the special case where the mutation factor is set to zero, the mutation operator becomes a crossover operator. Figure (11.1) displays graphically the solution space and how the mutation vector V is realised. Note, the parameter vector X_{r_3} that currently resides outside of the solution space demonstrates how the mutation procedure allows for exploration of alternative solution spaces and therefore, how the algorithm is able to reliably converge to global minima.

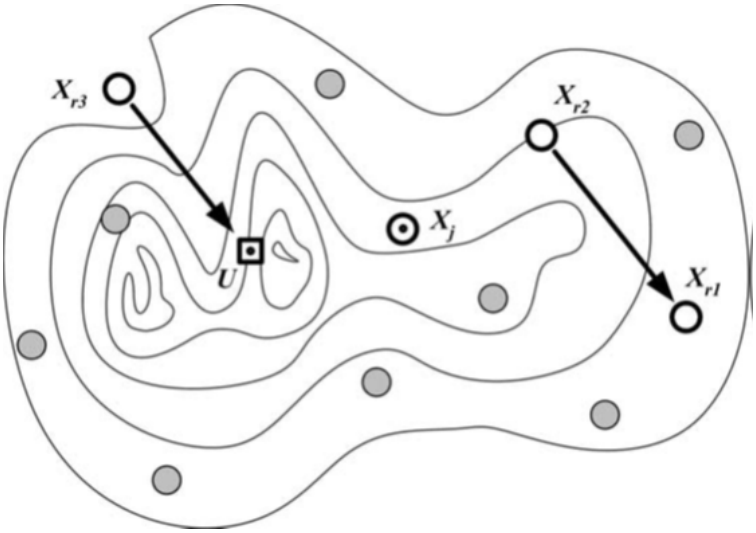


Figure 11.1: Creation of differential mutation vector. (Feoktistov [2006])

11.1.3.1.2 The recombination Recombination incorporates successful solutions from the previous generation. That is, according to a rule, we combine elements of the Target vector $X_{i,G}$ with elements of the Donor vector $V_{i,G}$ to create an offspring called the Trial vector $U_{i,G}$. In order to increase the diversity of the parameter vectors, elements of the Donor vector enter the Trial vector with probability CR. In the DE algorithm, each element of the Trial vector satisfies

$$U_{i,G}(j) = \begin{cases} V_{i,G}(j) \text{ for } j = n_r \mod dim, (n_r + 1) \mod dim, ..., (n_r + L - 1) \mod dim \\ X_{i,G}(j) \text{ for all other } j \in [0, .., NP - 1] \end{cases}$$

where $< n_r >_{dim} = n_r \mod dim$ is the modulo of n_r with modulus dim, dim is the dimension of the vector V (here $dim = N$), and the starting index n_r is a randomly chosen integer from the interval $[0, dim - 1]$. Hence, a certain sequence of the element of U is equal to the element of V, while the other elements get the original element of $X_{i,G}$. We only choose a subgroup of parameters for recombination, enhancing the search in parameter space. The integer L denotes the number of parameters that are going to be exchanged and is drawn from the interval $[1, dim]$ with probability

$$P(L > \nu) = (CR)^{\nu}, \nu > 0$$

The random decisions for both n_r and L are made anew at each new generation G. The term $CR \in [0, 1]$ is the crossover factor controlling the influence of the parent in the generation of the offspring. A higher value means less

Electronic copy available at: https://ssrn.com/abstract=3307566

influence from the parent. Generally, values of CR in the range $[0.8, 1]$ lead to good results (see Lampinen et al. [2004]). Most of the time, the mutation operator in Section (11.1.3.1.1) is sufficient and one can directly set the Trial vector equal to the Donor vector.

11.1.3.1.3 The selection Unlike the previous two procedures, the selection procedure is typically deterministic under Differential Evolution. The tournament selection only needs part of the whole population to calculate an individual selection probability where subgroups may contain two or more individuals. In the DE algorithm, the selection is deterministic between the parent and the child. The best of them remain in the next population. We compute the objective function with the original vector $X_{i,G}$ and the newly created vector $U_{i,G}$. If the value of the latter is smaller than that of the former, the new Target vector $X_{i,G+1}$ is set to $U_{i,G}$ otherwise $X_{i,G}$ is retained

$$X_{i,G+1} = \begin{cases} U_{i,G} \text{ if } H(U_{i,G}) \leqslant H(X_{i,G}) \, , i = 0, .., NP-1 \\ X_{i,G} \text{ otherwise} \end{cases}$$

This method, as applied in many Evolutionary Algorithms, ensures that the population fitness will always increase or remain constant through each generation. Mutation, recombination and selection continue until some stopping criterion is reached. The mutation-selection cycle is similar to the prediction-correction step in the EM algorithm or in the filtering problems. Figure (11.2) gives a graphical account of the tournament selection process.

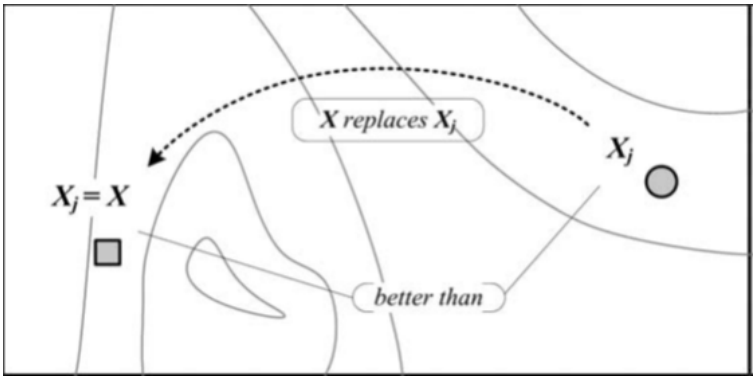


Figure 11.2: Tournament selection. (Feoktistov, [[2006]])

11.1.3.1.4 Simple convergence criteria We allow for different convergence criterion in such a way that if one of them is reached, the algorithm terminates. We let $f^* = f_{min}(G)$ be the fittest design in the population so far, and we define

$$f_{a,G} = \frac{1}{NP} \sum_{i=0}^{NP-1} f(X_{i,G})$$

as the average objective value at generation G. The variance of the objective value at generation G is given by

$$f_{v,G} = \frac{1}{NP} \sum_{i=0}^{NP-1} \left(f(X_{i,G}) - f_{a,G} \right)^2 = \frac{1}{NP} \sum_{i=0}^{NP-1} f^2(X_{i,G}) - f_{a,G}^2$$

Then, when the percentage difference between the average value and the best design reaches a specified small value ϵ_1

$$\frac{|f_{a,G} - f_{min}(G)|}{|f_{a,G}|} \times 100 \leqslant \epsilon_1$$

Electronic copy available at: https://ssrn.com/abstract=3307566

we terminate the algorithm. Also, we let $f_{min}(G-1)$ be the fittest design in the previous generation $(G-1)$, and consider as a criterion the difference

$$|f_{min}(G-1) - f_{min}(G)| < \epsilon_2$$

where ϵ_2 is user defined. In that case, the DE algorithm will continue until there is no appreciable improvement in the minimum fitness value or some predefined maximum number of iterations is reached.

11.1.3.2 Pseudocode

We now present the pseudo code of a standard DE algorithm.

```
Initialise vectors of the population NP
Evaluate the cost of each vector
    for i=0 to Gmax do
        repeat
            Select some distinct vectors randomly
            Perform mutation
            Perform recombination
            Perform selection
                if offspring is better than main parent then
                replace main parent in the population
                end if
        until population is completed
        Apply convergence criterions
    next i
```

11.1.3.3 The strategies

Over the years, Storn and Price as well as a large number of other authors made improvement to the DE model so that there are now many different DE models (see Price et al. [2005], Storn [2008]). These models vary in the type of recombination operator used as well as in the number and type of solutions used to calculate the mutation values. We are now going to list the main schemes perturbating the base vector for mutation.

11.1.3.3.1 Scheme DE1 To improve convergence on a set of optimisation problems using the Scheme DE4 introduced in Section (11.1.3.3.4), Storn and Price considered for each vector $X_{i,G}$ a trial vector V generated according to the rule

$$V = X_{r_1,G} + F\big(X_{r_2,G} - X_{r_3,G}\big)$$

where the integer r_l for $l = 1, 2, 3$ are chosen randomly in the interval $[0, NP-1]$ and are different from the running index i. Also, F is a scalar controlling the amplification of the differential variation $X_{r_2,G} - X_{r_3,G}$. We can slightly modify the scheme by writing

$$V = X_{r_1,G} + F\big(X_{r_2,G} + X_{r_3,G} - X_{r_4,G} - X_{r_5,G}\big)$$

More generally, the base vector can be expressed as a linear combination of other distinct vectors, as in Equation (11.1.1). As a simple rule, the differential weight F is usually in the range $[0.5, 1]$ (see Lampinen et al. [2004]). The population size should be between $3N$ and $10N$ and generally we increase NP if misconvergence happens. In the case where NP is increased we should decrease F.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.1.3.3.2 Scheme DE2 Similarly to the Scheme DE1 in Section (11.1.3.3.1), the trial vector V is generated according to the rule

$$V = X_{i,G} + \lambda\big(X_{best,G} - X_{i,G}\big) + F\big(X_{r_2,G} - X_{r_3,G}\big)$$

where λ is a scalar controlling the amplification of the differential variation $X_{best,G} - X_{i,G}$. It enhances the greediness of the scheme by introducing the current best vector $X_{best,G}$. It is useful for non-critical objective functions, that is, when the global minimum is relatively easy to find. It gives a balance between robustness and fast convergence.

11.1.3.3.3 Scheme DE3 In the same spirit, Mezura-Montes et al. [2006] also modified the Scheme DE1 in Section (11.1.3.3.1) by incorporating information of the best solution as well as information of the current parent in the current population to define the new search direction.

$$V = X_{r_3,G} + \lambda\big(X_{best,G} - X_{r_2,G}\big) + F\big(X_{i,G} - X_{r_1,G}\big)$$

where λ is a scalar controlling the amplification of the differential variation $X_{best,G} - X_{r_2,G}$. This scheme has the same properties as the Scheme DE2 in Section (11.1.3.3.2).

11.1.3.3.4 Scheme DE4 The oldest strategy developed by Storn and Price [1997] is for the trial vector V to be generated according to the rule

$$V = X_{best,G} + F\big(X_{r_1,G} - X_{r_2,G}\big)$$

where the weight F is a scalar. However, in that setting they found several optimisation problems where misconvergence occurred. To improve the convergence, Price et al. [2005] proposed to perturb the differential weight F by introducing the Dither and Jitter schemes. In the former, Karaboga et al. [2004] randomised the weight as follow

$$\lambda_G = F_l + U_G(0,1)\big(F_u - F_l\big)$$

where F_l is a lower weight, F_u is an upper weight, and $U_G(0,1)$ is uniformly distributed in the range $[0,1]$ for each generation G. In the latter, the random weight is picked for each element $j = 0, ..., D-1$ of the vector being updated

$$\lambda_j = F \times \big[1 + \delta \times (U_j(0,1) - \frac{1}{2})\big]$$

where δ determines the scale of perturbation. It must be small, and is usually set to $\delta = 0.0001$. For example, we can consider the strategy

$$V = X_{best,G} + \lambda_j\big(X_{r_1,G} - X_{r_2,G}\big)$$

It is a jitter which add fluctuation to the random target. The jitter is the time variation of a periodic signal in electronics and telecommunications (swing dancer). It is tailored for small population sizes and fast convergence. We can also modify the scheme by doing

$$V = X_{best,G} + \lambda_j\big(X_{r_1,G} + X_{r_2,G} - X_{r_3,G} - X_{r_4,G}\big)$$

Going one step further, Storn [2000] combined the jitter scheme with the dither one, getting

$$\lambda_{j,G} = \Big(F_l + U_G(0,1)\big(F_u - F_l\big)\Big)\big[1 + \delta \times (U_j(0,1) - \frac{1}{2})\big]$$

Electronic copy available at: https://ssrn.com/abstract=3307566

11.1.3.3.5 Scheme DE5 Das et al. [2005] improved the DE's convergence by applying the dither scheme to every difference vector. Similarly to the Scheme DE1 in Section (11.1.3.3.1), the trial vector V is generated according to the rule

$$V = X_{r_1,G} + \lambda_{d,i}\big(X_{r_2,G} - X_{r_3,G}\big)$$

where $\lambda_{d,i}$ is a computer dithering factor

$$\lambda_{d,i} = F + U_i(0,1) \times (1 - F)\ , i = 0, ..., NP - 1$$

also written

$$\lambda_{d,i} = F_l + U_i(0,1)\big(F_u - F_l\big)\ , i = 0, ..., NP - 1$$

where $U_i(0,1)$ is uniformly distributed in the range $[0,1]$. It is a per-vector dither, making the Scheme DE1 more robust. Note, as discussed in Section (11.1.3.3.4) we can also have

$$\lambda_{d,G} = F + U_G(0,1) \times (1 - F)$$

which is a per-generation dither factor. In that algorithm, choosing $F = 0.3$ is a good start. As explained by Pedersen [2010], the dither and jitter schemes are similar, except that the dither draws a random weight once for each agent-vector to be updated and the jitter draws a random weight for each element of that vector. To see this, we simply set $F_l = F \times (1 - \frac{\delta}{2})$ and $F_u = F \times (1 + \frac{\delta}{2})$. The dither can simply be rewritten as

$$\lambda_{d,i} \sim U_i(F_l, F_u)\ , i = 0, ..., NP - 1$$

where $U_i(F_l, F_u)$ is uniformly distributed in the range $[F_l, F_u]$, and the jitter can simply be rewritten as

$$\lambda_j \sim U_j(F \times (1 - \frac{\delta}{2}), F \times (1 + \frac{\delta}{2}))\ , j = 0, ..., D - 1$$

where $U_i(F \times (1 - \frac{\delta}{2}), F \times (1 + \frac{\delta}{2}))$ is uniformly distributed in the range $[F \times (1 - \frac{\delta}{2}), F \times (1 + \frac{\delta}{2})]$. Defining a midpoint F_{mid} and a range F_{range}, the dither becomes

$$\lambda_{d,i} \sim U_i(F_{mid} - F_{range}, F_{mid} + F_{range})\ , i = 0, ..., NP - 1$$

and the jitter becomes

$$\lambda_j \sim U_j(F_{mid} - F_{range}, F_{mid} + F_{range})\ , j = 0, ..., D - 1$$

We therefore need to select two parameters, F_{mid} and F_{range}, to determine the limits of perturbation for the weight F. Pedersen chose the midpoint in the interval $F_{mid} \in [0,2]$ and the range in the interval $F_{range} \in [0,3]$ allowing for negative differential weights to occur. Thus, perturbing behavioural parameters introduce new parameters in the form of the boundaries for the stochastic sampling ranges. Further, when the behavioural parameters are completely random, it takes statistically a lot of samples before finding an optimum.

11.1.3.3.6 Scheme DE6 A more complex algorithm than the Scheme DE1 in Section (11.1.3.3.1) is to introduce the choice between two strategies. In the either-or algorithm, the trial vector V is generated according to the rule

$$V = \begin{cases} \Big(X_{r_1,G} + F\big(X_{r_2,G} - X_{r_3,G}\big)\Big) I_{(\delta < \frac{1}{2})} \\ \Big(X_{r_1,G} + \frac{1}{2}(\lambda + 1)\big(X_{r_2,G} + X_{r_3,G} - 2X_{r_1,G}\big)\Big) I_{(\delta \geqslant \frac{1}{2})} \end{cases}$$

where δ is as above. It alternates between differential mutation and three-point recombination.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.1.3.3.7 Scheme DE7 Alternatively, we have the two possible strategies using an either-or algorithm. If we favor the r_1 event we have

$$V = \Big(X_{r_1,G} + F\big(X_{r_2,G} - X_{r_3,G}\big)\Big)I_{(\delta<P_e)}$$

while if we favor the best event we have

$$V = \Big(X_{best,G} + F\big(X_{r_2,G} - X_{r_3,G}\big)\Big)I_{(\delta<P_e)}$$

It alternates between differential mutation and doing nothing.

11.1.3.3.8 Scheme DE8 Still another more complex algorithm than the Scheme DE1 in Section (11.1.3.3.1) is to introduce two strategies V_l for $l = 1, 2$. We first build the Base vector $\widehat{V}_1$ as the linear combination of the two original base vector $X_{r_1,G}$ and $X_{best,G}$, getting

$$\widehat{V}_1 = \lambda_{M,G}X_{best,G} + \overline{\lambda}_{M,G}X_{r_1,G}$$

where $\overline{\lambda}_{M,G} = 1 - \lambda_{M,G}$. This time $\lambda_{M,G}$ is a Gaussian variable per-generation with mean μ and variance ξ to be defined. The second Base vector $\widehat{V}_2$ is generated according to the rule

$$\widehat{V}_2 = 2X_{best,G} - \widehat{V}_1 = (2 - \lambda_{M,G})X_{best,G} - \overline{\lambda}_{M,G}X_{r_1,G}$$

The two Trial vectors V_i for $i = 1, 2$ becomes

$$\begin{aligned} V_1 &= \widehat{V}_1 + F_b\big(X_{r_3,G} - X_{r_2,G}\big) \\ V_2 &= \widehat{V}_2 + F_b\big(X_{r_2,G} - X_{r_3,G}\big) = 2X_{best,G} - V_1 \end{aligned}$$

where F_b is a Gaussian variable with mean $\mu = 0$ and variance $\xi = F$. In the special case where $\mu = 2$ and $\xi = 0$ we get

$$\begin{aligned} \widehat{V}_1 &= 2X_{best,G} - X_{r_1,G} \\ \widehat{V}_2 &= X_{r_1,G} \end{aligned}$$

and the system simplifies to

$$\begin{aligned} V_1 &= 2X_{best,G} - X_{r_1,G} + F_b\big(X_{r_3,G} - X_{r_2,G}\big) \\ V_2 &= X_{r_1,G} + F_b\big(X_{r_2,G} - X_{r_3,G}\big) \end{aligned}$$

and V_2 recover a pseudo Secheme DE1 where $F_b \in [-F, F]$.

11.1.3.4 Improvements

The DE algorithm is found to be a powerful evolutionary algorithm for global optimisation in many real problems. As the DE algorithm performs mutation based on the distribution of the solutions in a given population, search directions and possible step sizes depend on the location of the individuals selected to calculate the mutation values. As a result, since the original article of Storn and Price [1995] many authors improved the DE model to increase the exploration and exploitation capabilities of the DE algorithm when solving optimisation problems. We are going to review a few changes to the DE algorithm which greatly improved the performances of our problem.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.1.3.4.1 The tuning parameters The tuning of the DE is mainly contolled by three variables: the number of vector (or individual) NP, the differential weight F, and the crossover factor CR. Finding bounds for their values has been a topic of intensive research. Zaharie [2002] studied theoretically the convergence of the DE by analysing the behavioural parameters of the DE models. She proved that the mutation scale factor F should never be smaller than F_{crit} defined as

$$F_{crit} = \sqrt{\frac{(1 - \frac{CR}{2})}{NP}}$$

Price et al. [2005] showed that only high values of CR guarantee the contour matching properties of DE. Further, only when $CR = 1$ is the mean number of function evaluation for an objective function and its rotated counterpart the same. In that setting, the DE is called rotationally invariant. As a rule of thumb (see Storn et al. [1997]), we get

- $F \in [0.5, 1.0]$
- $CR \in [0.8, 1.0]$
- $NP = 10D$

but they lack generality. Hence, the need to compute these parameters automatically.

11.1.3.4.2 Ageing The DE selection is based on local competition only. The number of children that may be produced to compete against the parent $X_{i,G}$ should be chosen sufficiently high so that a sufficient number of child will enter the new population. Otherwise, it would lead to survival of too many old population vectors that may induce stagnation. To prevent the vector $X_{i,G}$ from surviving indefinitely, Storn [1996] used the concept of ageing. One can define how many generations a population vector may survive before it has to be replaced due to excessive age. If the vector $X_{i,G}$ is younger than Num generations it remains unaltered otherwise it is replaced by the vector $X_{r_3,G}$ with $r_3 \neq i$ being a randomly chosen integer in $[0, NP - 1]$.

11.1.3.4.3 Constraints on parameters Commonly, we are searching for a solution to an optimisation problem between certain bounds. Given the parent vector $X_{i,G}$ for $i = 0, .., NP - 1$ we define upper and lower bounds for each initial parameters as

$$L(j) \leqslant X_{i,G_0}(j) \leqslant U(j)\ , j = 0, ..., D - 1$$

where G_0 is inception, and we randomly select the initial parameter values uniformly on the interval $[L(j), U(j)]$ as

$$X_{i,G_0}(j) = L(j) + U_j(0,1)\big(U(j) - L(j)\big)\ , j = 0, ..., D - 1$$

where $U_j(0,1)$ generates a random number in the range $[0,1]$ with a uniform distribution for each element j of the vector. Obviously, as the number of generation G increases, the DE algorithm will generate elements of the vector outside of the limits established (lower and upper) by an amount. Several alternatives exist for handling boundary constraints (see Onwubolu [2004]). Following Mezura-Montes et al. [2004a], this amount is substracted or added to the limit violated (reflecting barrier), in order to shift the value inside the limits. Should this cause the new value to violate the other bound, a random value will be generated, as when creating the initial parameter set. We can also set the element of the vector half way between the old position and the limit as follow

$$U_{i,G+1}(j) = \begin{cases} \frac{1}{2}(X_{i,G}(j) + U(j)) \text{ if } U_{i,G+1}(j) > U(j) \\ \frac{1}{2}(X_{i,G}(j) + L(j)) \text{ if } U_{i,G+1}(j) < L(j) \\ U_{i,G+1}(j) \text{ otherwise} \end{cases}$$

where $U_{i,G+1}(j)$ is the new Trial vector.

Electronic copy available at: https://ssrn.com/abstract=3307566

11.1.3.4.4 Convergence In order to accelerate the convergence process, when a child replaces its parent, Mezura-Montes et al. [2004a] copied its value both into the new generation and into the current generation. It allows the new child, which is a new and better solution, to be selected among the r_l solutions and create better solutions. Therefore, a promising solution does not need to wait for the next generation to share its genetic code. Similarly, to improve performance and to accelerate the convergence process, Storn [1996] explored the idea of allowing a solution to generate more than one offspring. Once a child is better than its parent, the multiple offspring generation ends. Following the same idea, Coello Coello and Mezura-Montes [2003] and then Mezura-Montes et al. [2006] allowed for each parent at each generation to generate $k > 0$ offspring. Among these newly generated solutions, the best of them is selected to compete against its parent, increasing the chances to generate fitter offspring.

11.1.3.4.5 Self-adaptive parameters It was proved that key control parameters in the DE algorithm, such as the crossover CR and the weight applied to random differential F, should be altered in the evolution process itself (see Liu et al. [2002]). These parameters can be adjusted by using heuristic rules, or they can be self-adapted (see Liu et al. [2005], Brest et al. [2006], Balamurugan et al. [2007]). That is, the control parameters are not required to be pre-defined and can change during the evolution process. These control parameters are applied at the individual levels in the population, such that better values should lead to better individuals producing better offspring and hence better values. However, in general, the random change to the parameters are applied irrespective of the quality of the current parameters. Noman et al. [2011] proposed to preserve better parameter choices, while changing the non-productive ones. We first describe the algorithm proposed by Brest et al. [2006] (jDE), and then introduce the adaptative algorithm of Noman et al. [2011] (aDE). The parameter F is a scaling factor controlling the amplification of the difference between two individuals to avoid search stagnation. At generation $G = 1$ the amplification factor $F_{i,G}$ for the ith individual ($i = 0, ..., NP - 1$) is generated randomly in the range $[0.1, 1.0]$. Then, at the next generations the control parameter is given by

$$F_{i,G+1} = \begin{cases} F_L + r_1 \times F_U \text{ if } r_2 < \tau_1 \\ F_{i,G} \text{ otherwise} \end{cases}$$

where r_j, for $j = 1, 2$ are uniform random values in $[0, 1]$, $F_L = 0.1$, $F_U = 0.9$ and τ_1 represent the probability to adjust the parameter F. Using the notation in Section (11.1.3.3.5), we can rewrite the scaling factor as

$$F_{i,G+1} = \begin{cases} U_i(F_l, F_u) \text{ if } r_2 < \tau_1 \\ F_{i,G} \text{ otherwise} \end{cases}$$

where $F_l = 0.1$ and $F_u = F_L + F_U = 1.0$. The only difference with the dither is the fact that the randomness of the differential weight depends on a probability of adjustment. According to Feoktistiv [2006], at the beginning of the evolution procedure, the mutation step length, and hence $F_{i,G}$, should be large, as individuals are far away from each other and the procedure could benefit from exploring beyond the current solution space. Since the individuals of a generation converge for subsequent generations, the step length, and hence $F_{i,G}$ should become smaller to allow a more concentrated search around the successful solution space. Thus, by applying custom parameters at individual levels, better values in the population lead to better individuals producing better Donor vectors and so on. Similarly, we can extend the crossover parameter CR as follow

$$CR_{i,G+1} = \begin{cases} r_3 \text{ if } r_4 < \tau_2 \\ CR_{i,G} \text{ otherwise} \end{cases}$$

where r_j, for $j = 3, 4$ are uniform random values in $[0, 1]$ and τ_2 represent the probability to adjust the parameter CR. The new parameter CR takes random values in the range $[0, 1]$. Since $F_L = 0.1$, $F_U = 0.9$, Brest et al. [2006] proposed to set $\tau_1 = \tau_2 = 0.1$ such that F takes random values in the range $[0.1, 1]$. Note, both $F_{k,G+1}$ and $CR_{k,G+1}$ are obtained before the mutation is performed in order to influence the mutation, crossover, and selection of the new vector $X_{i,G+1}$. Given that random adjustment can only be good when the parameter setting is not suitable, Noman et al. [2011] proposed to compare the fitness of the offspring $f(U_G)$ with the average fitness value of the current generation, $f_{a,G}$. Then, the choice of the amplification factor F in offspring U_G for the parent $X_{i,G}$ is given by

Electronic copy available at: https://ssrn.com/abstract=3307566

$$F_G^{child} = \begin{cases} F_{i,G} \text{ if } f(U_G) < f_{a,G} \\ r_1(0.1, 1.0) \text{ otherwise} \end{cases}$$

and that of the crossover parameter is given by

$$CR_G^{child} = \begin{cases} CR_{i,G} \text{ if } f(U_G) < f_{a,G} \\ r_2(0.1, 1.0) \text{ otherwise} \end{cases}$$

where $r_j(a, b)$ are uniform random number in the range $[a, b]$. Initially, the parameters F and CR are created randomly for each individual. Note, the objective vector part of the offspring is created by using the original mutation and crossover values of the DE.

***Remark* 11.1.1** *Again, perturbing behavioural parameters introduce new parameters in the form of the boundaries for the stochastic sampling ranges and the adjustment probabilities.*

11.1.3.4.6 Selection Santana-Quintero et al. [2005] maintained two different populations (primary and secondary) according to some criteria and considered two selection mechanisms that are activated based on the total number of generation G_{max} and the parameter $sel_2 \in [0.2, 1]$ which regulates the selection pressure. That is

$$\text{Type of selection } = \begin{cases} \text{Random if } G < (sel_2 * G_{max}) \\ \text{Elitist otherwise} \end{cases}$$

a random selection is first adopted followed by an elitist selection. In the random selection, three different parents are randomly selected from the primary population while in the elitist selection they are selected from the secondary one. In both selections, a single parent is selected as a reference so that all the parents of the main population will be reference parents only once during the generating process.

11.1.3.5 Convergence criteria revised

When applying a genetic algorithm to solve some NP-hard optimisation problem (the optimum is unkown), it is usually infeasible to compute the optimal solution of the problem. Still, we need to determine whether or not the algorithm has converged to some optimum. Since a genetic algorithm is said to converge when there is no significant improvement in the values of fitness of the population from one generation to the next, there is no defined difference between stopping criteria and convergence criteria. That is, the stopping criterion provides the user a guideline in stopping the algorithm with an acceptable solution close to the optimal solution. While, mathematically, that closeness may be judged in various ways, termination criteria should avoid needless computation and prevent premature termination. Thus, stopping criteria should account for

- Reliability guarantees termination within a finite time.
- Performace guarantees no premature termination and no needless computation.

Stopping criteria are generally based on time or fitness value. The simpliest termination criteria, called exhaustion-based criteria, is to select a maximum number of generations of evaluation functions, or, a maximal time budget (absolute time, CPU time). Alternatively, we can terminate the search when the best objective value $f^* = f_{min}$ reaches or surpasses a bound, $f^* \leqslant f_{lim}$. Usually, if there is no change in the best fitness value for K consecutive iterations, the algorithm is terminated. This method works well if the optimum or a lower bound is known. Some authors proposed tight bounds on the number of iterations required to achieve a level of confidence to guarantee that a Genetic Algorithm has seen all strings (see Aytug et al. [1996]). Giggs et al. [2006] empirically studied a way to determine the maximum number of generations. However, determining the optimum time, or, finding a lower bound is a challenge. Thus, stopping criteria should contain the advantage of reacting adaptively to the state of the optimisation run.

Electronic copy available at: https://ssrn.com/abstract=3307566

The criteria based on objective function values use the underlying fitness function values to calculate auxiliary values as a measure of the state of the convergence of the GA. For instance,

- improvement-based criteria monitor the improvement of the best objective function value (ImpBest) (or its average ImpAvg) along the optimisation process, and stops when it falls below a user-defined threshold for a given number of generations.
- movement-based criteria monitor the distances between the population members in successive iterations (see Schwefel [1995]). The movement in the population can be calculated with respect to the average objective function value (MovObj), or with respect to positions (MovPar). Termination occurs when it is below a threshold for a given number of generations.
- distribution-based criteria monitor the distances of the population members at each iteration (see Zielinski et al. [2008]). It is assumed that all individuals converge to the optimum, such that convergence is reached when they are close to each other. MaxDist is when the maximum distance from every vector to the best population vector is below a threshold.
- combined criteria use several criteria in combination.

It is understood that the algorithm often focuses on global optimisation at the beginning, leading to large movements between population members in successive iterations, and that at the final stages of the optimisation process, the population generally converges to one point. There are different ways of measuring these distances, such as the standard deviation of positions to all, or part, of the population members, or the distance between the individuals with the best and worst objective function value. In any case, when the distance falls below a user-defined threshold for a given number of iterations, the algorithm terminates. For example, the Running Mean is the difference between the current best objective value $f_{min}(G)$, at generation G, and the average of the best objective value

$$f_{a,min}(n) = \frac{1}{G_{last}} \sum_{i=0}^{n} f_{min}(G_i) \, , G_i = G - i\Delta G$$

over a period of time $G_{last} = n\Delta G$, where ΔG is one generation. It must be less or equal to a given threshold, that is,

$$|f_{min}(G) - f_{a,min}(n)| \leqslant \epsilon$$

The Best-Worst is the difference between the best objective value $f_{min}(G)$ and the worst one $f_{max}(G)$ at generation G. At least $p\%$ of the individuals must be feasible, and it must be less or equal to a given threshold, $|f_{min}(G) - f_{max}(G)| \leqslant \epsilon$. We can also consider relative termination criterions such as $\frac{f_{min}}{f_{a,G}} \leqslant \epsilon$, or letting d_{ij} be the sum of all normalised distance between all individuals of the current generation, the ratio $\frac{d_{ij}}{K_{max}} \leqslant \epsilon$. The latter values the spacial speading of individuals of the current generation in the search space (normalised Euclidian distances $\frac{d_{ij}}{d}$ where d is the length of diagonal of the search space). Jain et al. [2001] proposed the Clus Term, combining information from the objective values and the distribution of individuals in the search space. They perform a cluster analysis (single linkage method) of the fittest individuals and determine the total amount $N(t)$ of of individuals in clusters. The search is terminated when the change of the average of $N(t)$ is equal or less than ϵ. Analysis of stopping criteria reacting adaptively to the state of an optimisation were perfomed (see Zielinski et al. [2005], Zielinski et al. [2007]).

Generally, EAs are stopped or terminated when the variance of fitness values of all the strings in the current population is less than a predefined threshold ϵ. It is assumed that after significantly many iterations the fitness values of the strings present in the population are all close to each other (the population becomes homogeneous), thereby making the variance of the fitness values close to 0. Thus ϵ should be chosen close to zero. Further, one should select a significant number of iterations from which the fitness values will be considered in calculating the variance so that the algorithm gets enough opportunity to yield improved (better) solution. However, this is not correct since

Electronic copy available at: https://ssrn.com/abstract=3307566

- in elitist model, or other GAs, only the best string is preserved.
- any population containing an optimal string is sufficient for the convergence of the algorithm.
- there is a positive probability of obtaining a population after infinitely many iterations with exactly one optimal string and others are being not optimal.

A lot more iterations often occur after the global optimum has been reached, or nearly reached, to improve other inferior individuals. However, only the best objective function value is important, rather than the convergence of the whole population. Since at the beginning, global search dominates the optimisation algorithm, while at final stages, the algorithm focuses on local optimisation, Liu et al. [2009] proposed a combination of global and local methods. They monitor the average improvement of the whole population in the former, and they monitor the best objective function value in the latter. Bhandari et al. [2012] established theoretically that the variance of the best fitness values obtained in the iterations can be considered as a measure to decide the termination criterion of a GA with elitist model (EGA). The proposed criterion uses only the fitness function values and takes into account the inherent properties of the objective function. We let $f_{min}(i)$ be the best fitness function value obtained at the end of ith iteration, such that $f_{min}(1) \geqslant f_{min}(2) \geqslant ... \geqslant f_{min}(n) \geqslant ... \geqslant F_1$, where F_1 is the global optimal value of the fitness function (F_i denotes the ith lowest fitness function value). Then we get the statistical mean and variance of the best fitness values obtained up to the nth iteration as

$$\begin{aligned} f_{min,1}(n) &= \frac{1}{n}\sum_{i=1}^{n} f_{min}(i) \\ b(n) &= Var(f_{min}(n)) = \frac{1}{n}\sum_{i=1}^{n}\left(f_{min}(i) - f_{min,1}(n)\right)^2 = f_{min,2}(n) - f_{min,1}^2(n) \end{aligned}$$

where $f_{min,2}(n) = \frac{1}{n}\sum_{i=1}^{n} f_{min}^2(i)$. The variance can be iteratively calculated as follow

$$b_{n+1} = \frac{1}{n+1}\left((nf_{min,2}(n) + f_{min}^2(n+1)) - \left(f_{min,1}(n) + f_{min}(n+1)\right)^2\right)$$

such that only $f_{min,1}(n)$ and $f_{min,2}(n)$ at step n are required to keep in memory when computing the variance at step $(n+1)$. Alternatively, following Equation (14.5.6), we can write recursively the sample mean as

$$f_{min,1}(n+1) = f_{min,1}(n) + \frac{f_{min}(n+1) - f_{min,1}(n)}{n+1}$$

and following Equation (14.5.7), the sample variance becomes

$$b(n+1) = b(n) + f_{min,1}^2(n) - f_{min,1}^2(n+1) + \frac{f_{min}^2(n+1) - b(n) - f_{min,1}^2(n)}{n+1}$$

So far, the variance based criterion is not scale invariant, meaning it is sensitive to transformations of the fitness function [1]. One can easily avoid the impact of the scaling effect by a simple transformation of the fitness function, such as

$$g(x) = \frac{f(x)}{f_{min}^1}$$

where f_{min}^1 is the minimum value of the fitness function obtained in the first iteration. If we let $b_n(g)$ be the variance of the best fitness values obtained up to the nth iteration for the function $g(x)$, then we get

[1] $g(x) = k \times f(x)$ where k is a constant.

Electronic copy available at: https://ssrn.com/abstract=3307566

$$b_n(g) = \frac{1}{n(f^1_{min})^2}\sum_{i=1}^{n}\left(f_{min}(i) - f_{min,1}(n)\right)^2 = \frac{1}{(f^1_{min})^2}b_n(f)$$

such that assuming the tolerance level ϵ_f as the value ϵ for the function f is equivalent to assuming $\epsilon_f \times (f^1_{min})^2$ as the value of ϵ for the function g. It implies that the user has to adjust the value of ϵ for the applied transformation. Note, we now need to use Equation (14.5.8) to obtain recursively the sample mean, and Equation (14.5.9) to obtain recursively the sample variance. In that setting, the GA is stopped or terminated after N iterations when the variance of the best fitness values obtained so far is bounded. That is, $b_N < \epsilon$, where $\epsilon > 0$ is a user defined small quantity corresponding to the difference between the fitness value of the best solution obtained so far and the global optimal solution.

11.2 Handling the constraints

11.2.1 Describing the problem

We saw above that EAs in general, and DE in particular, lacked a mechanism to deal with the constraints of the problems. Recently, various academics worked on solving that problem, and one of the most popular constraint handling mechanisms was proposed by Deb [2000] on genetic algorithms (GAs) who used the three feasibility rules. This algorithm generates feasible individuals, while maintaining a reasonable ratio between feasible and infeasible members in a population, allowing for a sparse feasible domain. Several studies assessed the performances of these rules on a large number of test functions, and very good results were found (see Zielinski et al. [2006], Zhang et al. [2012]). Before reviewing his method and showing how it was improved, we recall that Michalewicz [1995] discussed different constraint handling methods used in GAs and classified them in five categories

- methods based on preserving feasibility of solutions, that is, we use a search operator that maintains the feasibility of solutions
- methods based on penalty functions
- methods making distinction between feasible and infeasible solutions using different search operators for handling infeasible and feasible solutions
- methods based on decoders using an indirect representation scheme which carries instructions for constructing feasible solutions
- hybrid methods where evolutionary methods are combined with heuristic rules or classical constrained search methods

In a single-objective optimisation problem, the traditional approach for handling constraints is the penalty function method. The fitness of a candidate is based on a scale function F which is a weighted sum of the objective function value and the amount of design constraint violation

$$F(X) = f_1(X) + \left(\sum_{k=1}^{p}\omega_k \max\left(g_k(X), 0\right) + \sum_{k=p+1}^{q}\omega_k|h_k(X)|\right)$$

where ω_k are positive penalty function coefficients and such that the kth constraint $g_k(.)$ and $h_k(.)$ should be normalised. This method requires a careful tuning of the coefficients ω_k to obtain satisfactory design, that is a balance between the objective function and the constraints but also between the constraints themselves. Kusakci et al. [2012] presented a literature review summerising the constraint handling techniques for constrained optimisation problems (COPs).

Electronic copy available at: https://ssrn.com/abstract=3307566

11.2.2 Defining the feasibility rules

To overcome this problem, Deb [2000] proposed a penalty function approach based on the non-dominance concept, ranking candidates using the definition of domination between two candidates.

Definition 11.2.1 *A solution i is said to dominate a solution j if both of the following conditions are true*

1. *solution i is no worse than solution j in all objective*

$$\forall f_m(X_i) \leqslant f_m(X_j)$$

2. *solution i is strictly better than solution j in at least one objective*

$$\exists f_m(X_i) < f_m(X_j)$$

The constrained domination approach ranks candidates according to the following definition

Definition 11.2.2 *A solution i is said to constrained-dominate a solution j if any of the following conditions is true*

1. *solutions i and j are feasible and solution i dominates solution j.*
2. *solution i is feasible and solution j is not.*
3. *both solutions i and j are infeasible but solution i has a smaller constraint violation.*

He let the fitness function be

$$F(X) = \begin{cases} f(X) \text{ if } g_k(X) \leqslant 0 \ \forall k = 1, 2, .. \\ f_{max} + TACV \text{ otherwise} \end{cases}$$

where f_{max} is the objective value with the worst feasible solution in the population and (TACV) is the total amount of constraint violation

$$TACV = \sum_{k=1}^{p+q} \max\left(g_k(X), 0\right)$$

Therefore, solutions are never directly compared in terms of both objective function and constraint violation information. However, the high selection pressure generated by tournament selection will induce the use of additional procedure to preserve diversity in the population such as niching or sharing. Clearly, there is no tuning of the penalty function coefficients when the number of constraint is one. But, when multiple constraints are considered some considerations must be taken to relate constraints together. One way forward is to normalise the constraints such that every constraint has the same contribution to the comparing value as was done by Landa Becerra et al. [2006]. Letting $g_{max}(k)$ be the largest violation of the constraint $\max\left(g_k(X), 0\right)$ found so far, we define the new TACV as

$$NTACV = \sum_{k=1}^{p} \frac{\max\left(g_k(X), 0\right)}{g_{max}(k)}$$

11.2.3 Improving the feasibility rules

Again, many different approaches were proposed, for instance Coello Coello [2000] modified the definition of the constrained domination approach given in Definition (11.2.2) such that if the individuals are infeasible he compares the number of constraints violated first and only in the case of a tie would he use the total amount of constraint violation in the definition, getting

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 11.2.3 *A solution i is said to constrained-dominate a solution j if any of the following conditions is true*

1. *solutions i and j are feasible and solution i dominates solution j.*
2. *solution i is feasible and solution j is not.*
3. *both solutions i and j are infeasible but solution i violates less number of constraints than solution j.*
4. *both solutions i and j are infeasible and violating the same number of constraints but solution i has a smaller TACV than solution j.*

In that setting, the fitness of an infeasible solution not only depends the amount of constraint violation, but also on the population of solutions at hand. However, this technique may not be very efficient when the degrees of violation of constraints $g_k(X)$ are significantly different because the TACV is a single value. Alternatively, Coello Coello et al. [2002] handled constraints as additional objective functions and used the non-dominance concept (on objective functions) in Definition (11.2.1) to rank candidates. As a result, it required solving the objective function a large number of time. Going one step further, Oyama et al. [2005] introduced dominance in constraint space.

Definition 11.2.4 *A solution i is said to dominate a solution j in constraint space if both of the following conditions are true*

1. *solution i is no worse than solution j in all constraints*

$$\forall g_k(X_i) \leqslant g_k(X_j)$$

2. *solution i is strictly better than solution j in at least one constraint*

$$\exists g_k(X_i) < g_k(X_j)$$

Introducing the idea of non-dominance concept to the constraint function space, their proposed constraint-handling method is

Definition 11.2.5 *A solution i is said to constrained-dominate a solution j if any of the following conditions is true*

1. *solutions i and j are feasible and solution i dominates solution j in objective function space.*
2. *solution i is feasible and solution j is not.*
3. *both solutions i and j are infeasible but solution i dominates solution j in constraint space.*

In that setting, any non-dominance ranking can be applied to feasible designs and infeasible designs separately. As a result, in a single-objective constrained optimisation problem, Bloch [2010] modified the dominance-based tournament selection of Coello and Mezura with the non-dominance concept of Oyama et al., getting

Definition 11.2.6 *The new dominance-based tournament selection is*

1. *if solutions i and j are both feasible and solution i dominates in objective function solution j then solution i wins.*
2. *if solution i is feasible and solution j is not, solution i wins.*
3. *if solutions i and j are both infeasible and solution i dominates in constraint space solution j then solution i wins.*
4. *if solutions i and j are infeasible and non-dominated in constraint space, if solution i violates less number of constraints than solution j then solution i wins.*
5. *if solutions i and j are both infeasible, non-dominated in constraint space and violating the same number of constraints but solution i has a smaller TACV than solution j then solution i wins.*

Electronic copy available at: https://ssrn.com/abstract=3307566

11.2.4 Handling diversity

In order to explore new regions of the search space and to avoid premature convergence, a set of feasibility rules coupled with a diversity mechanism is proposed by Mezura-Montes et al. [2006]. It maintains besides competitive feasible solutions some solutions with a promising objective function value allowing for the DE to reach optimum solutions located in the boundary of the feasible region of the search space. Given a parameter S_r one can choose to select between parent and child based either only on the objective function value or on feasibility. As more exploration of the search space is required at the beginning of the process, the parameter S_r will be decreasing in a linear fashion with respect to the number of generation. Given an initial value $S_{r,0}$ and a terminal value $S_{r,\infty}$, the adjustment of the parameter is

$$S_{r,G+1} = \begin{cases} S_{r,G} - \Delta_{S_r} \text{ if } G > G_{max} \\ S_{r,\infty} \text{ otherwise} \end{cases}$$

where $\Delta_{S_r} = \frac{S_{r,0} - S_{r,\infty}}{G_{max}}$. After some generations, assuming that promising areas of the feasible region have been reached, we focus on keeping the feasible solutions found discarding the infeasible ones. In that setting, infeasible solutions with good objective function values will have a significant probability of being selected which will slowly decrease as the number of generation increases.

In some problems, it does not make sense to use the infeasible individual unless they are very close to the boundary between the feasible and infeasible region. This is the case of the calibration problem given in Section (1) where the constraints ensure that prices do not violate the AAO rules. To circumvent this issue, Mezura-Montes et al. [2004a] proposed that at each generation the best infeasible solution with lowest amount of constraint violation both in the parents (μ) and the children (λ) population will survive. Either of them being chosen with an appropriate probability, it will allows for infeasible solutions close to the boundary to recombine with feasible solutions. The pseudo code for introducing diversity in the DE algorithm is

```
if flip(S_r) then
   Select the best infeasible individual from the children population
else
   Select the best individual based on five selection criteria
end if
```

11.3 The proposed algorithm

Using the improvements in Section (11.1.3.4) and applying the Definition (11.2.6) on dominance-based tournament selection to handle the constraints, the pseudo code for the DE algorithm with constraints becomes

```
Begin
 G = 0 and Age_i,G = 0 ∀ i, i = 1,..,NP
Create a random initial population X_i,G ∀ i, i = 1,..,NP
Evaluate f(X_i,G) ∀ i, i = 1,..,NP
while niter < max_iter and G < Gmax do
    fmin_old = fmin
   for i = 1 to NP do
      for k = 1 to N_K do
         Select randomly r_1 ≠ r_2 ≠ r_3 ≠ i
          j_r = U(1,N)
         for j = 1 to N do
            if U(0,1) < CR or j = j_r then
```


Electronic copy available at: https://ssrn.com/abstract=3307566

```
                U_i,G(j) = X_r_3,G(j) + F (X_r_1,G(j) − X_r_2,G(j))
            else
                U_i,G(j) = X_i,G(j)
            end if
         end for
         if k > 1 then
            if U_i,G(j) is better than U_i_best,G(j) based on five selection
            criteria then
                U_i_best,G(j) = U_i,G(j)
         else
             U_i_best,G(j) = U_i,G(j)
      end of for
      Apply diversity :
      if flip(S_r) then
         if f(U_i_best,G) ⩽ f(X_i,G) then
             X_i,G+1 = X_i,G = U_i_best,G
         else
             X_i,G+1 = X_i,G
         end if
      else
         if U_i_best,G is better than X_i,G based on five selection criteria then
             X_i,G+1 = X_i,G = U_i_best,G
         else
            if Age_i,G < N_A or i = i_best then
                X_i,G+1 = X_i,G
            else
               Select randomly r_4 ≠ i
                X_i,G+1 = X_r_4,G
                Age_i,G = 0
            end if
         end if
      end if
      if fmin > f(X_i,G)
          fmin = f(X_i,G)
         i_best = i
      end if
   end for
   Apply convergence criterions :
   if fmin_old − f_min < precision
      niter = niter + 1
   else
      niter = 0
   end if
    G = G + 1
end while
End
```


Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 12

Solving optimisation problems with NNs

12.1 Hybrid intelligent modelling

Data mining (see Section (2.2)) and more recent machine-learning methodologies (see Chapter (4)) provide a range of general techniques for the classification, prediction, and optimisation of structured and unstructured data. All of these methods require the use of quantitative optimisation techniques known as stochastic optimisation algorithms, such as combinatorial optimisation, simulated annealing (SA), genetic algorithms (GA), or reinforced learning. We find GA, ANN and SVM algorithms among the most popular classifiers. The artificial neural network (ANN) is the most widely used technique for classification and prediction. It is an optimisation task where the result is to find optimal weight and bias set of the network that reduce an error function. It can be formulated as the minimisation of an error function in the space of connection weights (see Section (8.1)). However, ANN has many disadvantages and some of its limitations are follows:

- ANN has long trained time
- High computational cost
- Adjustment of weight is difficult
- Output quality of an ANN may be unpredictable

To overcome these disadvantages Neural Networks can be combined with another technique. One way forward is to combine ANN with another algorithm such as evolutionary computing tools that can take care of a specific problem. Hybridisation is a technique which combines two or more classifiers to improve the performance of the classifier. Many researchers proposed hybrid technique with an evolutionary algorithm (EA) (see Section (11.1)) to improve the performance of the classifier. For instance, GAs has been hybridised with many other classification algorithms (see Montana et al. [1989]). Alternatively, Ilonen et al. [2003] applied Differential Evolution (DE) (see Section (11.1.3)) to train the weights of neural networks, obtaining convergence to a global minimum.

12.1.1 ANN plus GA in finance

12.1.1.1 An overview

The underlying principles of genetic algorithm (GA) (see Section (11.1.2.5)) are to generate an initial population of chromosomes (search solutions) and then use selection and recombination operators to generate a new, more effective population which eventually will have the fittest chromosome (optimal value) among them. Some examples of combition between ANN and GA are Whitley et al. [1990] who used GA to optimise weighted connections and find a good architecture for neural network connections. Kim [2006] proposed a hybrid model of ANN with GA that performs

Electronic copy available at: https://ssrn.com/abstract=3307566

instance selection to reduce dimensionality of data. The genetic algorithm is used to optimise the connection weights between layers of neural network and for selection of relevant instance. Due to the selected instances, the learning time is reduced and prediction performance is enhanced. Bai et al. [2006] presented an effective hybrid system on rough set and neural network. This classifier is used for managing the large document that is growing on the internet. The system creates clusters for organising the documents. The rough set is used for feature reduction and Neural Network is used for classification. Vivekanandan et al. [2010] built a rule-based classifier by using the genetic algorithm. The proposed work tries to reduce the learning time by incremental learning. To do so the classifier reduces the cost of learning and thus making it scalable for large data sets. Large researches have been performed in applying genetic algorithms for classification purpose. For instance, Bhardwaj et al. [2015] proposed a model called GONN which hybridise genetic programming with neural network for multi class classification problem. The model simultaneously deals with structure and weight of the neural network, bringing more diversity of the GP population and reaching a solution faster with more generalised solutions. GAs has been also widely used for the discovery of classification rules.

12.1.1.2 Some examples

12.1.1.2.1 Feature selection Sangwan et al. [2015] proposed an integrated ANN and GA for predictive modelling and optimization of turning parameters to minimize surface roughness. Following the same approach, Inthachot et al. [2016] used GA to solve a feature selection problem to find effective subsets of input into ANN. Considering four input variables for each technical indicator, to speed up computation time, they used GA to devise a small number of effective subsets of input variables (see Figure (12.1)). Accuracy is used to determine chromosome selection as well as to measure the performance of the prediction model.

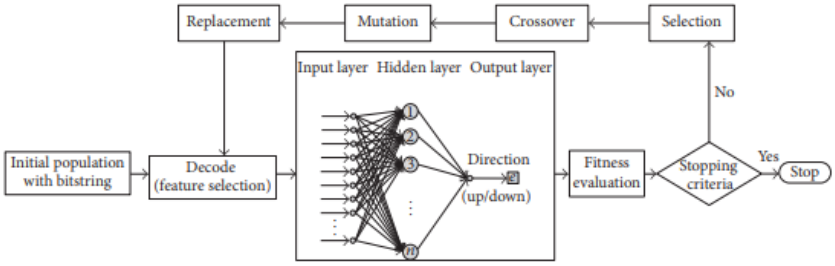


Figure 12.1: Steps of operation of ANN and GA hybrid intelligence

12.1.1.2.2 Weight selection Karimi et al. [2012] used GA to find a set of weights for connections to each node in an ANN model and determine correlation of density in nanofluids. Qiu et al. [2016] proposed to determine the optimal set of weights and biases to enhance the accuracy of an ANN model (backpropagation algorithm) by using genetic algorithms (GA). They reported improved prediction accuracy of the direction of stock price index movement with their optimised ANN model. They considered a three-layer ANN model where the output layer consists of only one neuron that represents the predicted direction of the daily stock market index. The signals are generated on a daily basis at time interval $\delta = 1$ day and the forecasting period is $d = 1$ day. GA technique is used to optimise the (initial) weights and biases of the ANN model. Then, the ANN model is trained by the BP algorithm using the determined weights and bias values. The steps are as follow:

- The data is first normalised (see Equation (8.3.16)), then all the weights and biases are encoded in a string and the initial population is generated. Each solution from the GA is called chromosome (or individual) describing the ANN with a certain set of weights and bias values.
- Train the ANN model using the BP algorithm and then evaluate each chromosome of the current population using a fitness function based on the MSE.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Rank all the individuals using the fitness proportion method and select the individuals with a higher fitness value to pass on to the next generation directly.
- Apply crossover and mutation to the current population and create new chromosomes. Evaluate the fitness value of the new chromosomes and insert these new chromosomes into the population to replace the worse individuals of the current population.
- Repeat the previous steps until the stop criterion is satisfied.

12.1.2 ANN plus DE in finance

The training (error) of artificial neural networks (ANNs) depends on the adaptation of free network parameters, such as the weight values and the bias. Thus, it is an optimisation task where the result is to find optimal weight and bias set of the network that reduce an error function. It can be formulated as the minimisation of an error function in the space of connection weights (see Section (8.1.1)). The methods for training ANNs are backpropagation (BP), Levenberg-Marquadt (LM), Quasi-Newton (QN). In general, the error backpropagation method (EBP), based on gradient method, is preferred. However, the objective function describing the artificial neural networks training problem is a multi-modal function, so that the algorithms based on gradient methods can easily be stuck in local extremes. To avoid this problem one can use a global search optimisation, such as simulating annealing (SA) algorithm (see Aarst et al. [1989]), Particle Swarm Optimization (PSO) and Evolutionary Algorithm (EA) (see Michalewicz [1996]). For instance, Junyou [2007] used PSO-trained neural networks to forecast the stock price.
We have seen above that when training the ANN by using GA method (GANN), the weight coefficients of neural network are encoded in chromosome, and selection, cross-over and mutation operators are used to minimise the error. However, GANN suffers from early convergence. Even though GA has diversity in its population, it lacks of convergence speed towards global optimia. On the other hand, PSO has faster convergence speed than that of GA but it lacks of diversity in population.

We focus on differential evolution (DE) (see Section (11.1.3)), which has the following advantages: the possibility of finding the global minimum of a multi-modal function regardless of initial values of its parameters, quick convergence and a small number of parameters to set up at the start of the algorithm operation.
Ilonen et al. [2003] applied Differential Evolution to train the weights of neural networks, obtaining convergence to a global minimum but in a very long time. To do so, the weights of all neurons are stored in a real valued solution vector X. The algorithm is used to minimise the sum of squared errors (SSE) or a similar criterion function. The evaluation of this function requires iterating through all elements of the training set T and summing all the partial results (squared errors in the case of SSE) obtained for all the elements of T. Note, different choices of behavioural parameters cause it to perform worse or better than BP on particular problems and the selection of good parameters is a challenge. One way forward is to try and make the DE parameters automatically adapt to new problems during optimisation, hence alleviating the need for the practitioner to select the parameters by hand. Since some meta parameters are kept fix throughout the entire evolution process, one must tune value of these control parameters. One approach is to use self-adaptive strategy for controlling these parameters (see Brest et al. [2006]). Slowik et al. [2008] applied DE technique to train a feed-forward flat ANN to classification of parity-p problem by introducing the adaptive selection of control parameters to the algorithm. The method proposed is called DE-ANNT (Differential Evolution ? Artificial Neural Network Training). As a result, only one parameter is set at the start of proposed algorithm. Similarly, Bui et al. [2015] use self-adaptive strategy to control the DE parameters. Rather than considering self-adaptive parameters, Pedersen et al. [2008b] proposed an overlaying optimisation method known as meta-optimisation.

Even though EA simultaneously process a population of problem solutions, it comes at the expense of very high computational complexity. To remedy this problem, Fan et al. [2003] introduced Trigonometric DE (TDE) algorithm and applied to train the ANN as a test case for their proposed algorithm. Bandurski et al. [2009] proposed to combine differential evolution algorithm with a gradient-based approach. This hybridisation comes in a number of ways, some of them are as follows:

Electronic copy available at: https://ssrn.com/abstract=3307566

1. One solution is for an EA to be used to locate a promising region of the weight space, and then use a gradient descent method to fine-tune the best solution (or all the solutions from the population) obtained by the EA (see Sherrod [2008]).

2. Another solution is to apply Lamarckian evolution (see Ross [1999]) to ANN and let the gradient descent procedure be incorporated into an EA as a new search operator. This operator is applied to the population members in each EA iteration, in addition to standard operators such as mutation and crossover (see Cortez et al. [2002]).

The authors applied the conjugate gradient (CG) algorithm (see Hestenes et al. [1952]) to each candidate solution $U_{i,G}$ (Trial vector in Appendix (11.1.3.1)) before the computation of its fitness. The number of CG iterations is set by the user and remains constant throughout the entire experiment. The results obtained were significantly faster than the original version of DE.
Similarly, to keep a reasonable balance between convergence speed and the capability of global search, Mingguange et al. [2009] combined the BP and DE algorithm to optimise the weights and threshold value adjustments of ANN.

Si et al. [2012] used DE with global and local neighbourhood based mutation (DEGL) algorithm to search the synaptic weight coefficients of neural network and to minimise the learning error in the error surface. It is a modification of DE where both global and local neighbourhood-based mutation operator is combined to create donor vector.

12.2 Solving constrained optimisation problems

In general, optimisation problems that can not be solved with an exact algorithm are solved by applying a global search optimisation such as genetic algorithms, particle swarm optimisation, simulated annealing, ant colony optimisation etc. Some authors proposed the use of neural networks (NN) to resolve optimisation problems in those cases where the use of linear programming or Lagrange multipliers (see Section (10.4)) is not feasible.

12.2.1 The method

12.2.1.1 A short review

Artificial neural networks (ANN) have been used to obtain solution of constrained optimisation problems (see Chapter (10)). For instance, Chua et al. [1984] developed the canonical non-linear programming circuit, using the Kuhn-Tucker conditions from mathematical programming theory (see Section (10.4)). Hopfield et al. [1985] [1987] published some results on how to go about using neural networks to solve optimisation problems. The Travelling Salesman Problem was the first problem formulated in terms of neural network. The authors modified the energy function of the network to make it equivalent to TSP objective and used Lagrange multipliers to penalise the violations of the constraints. The approach was then used to demonstrate how circuits of simple unit can solve hard problems. Tank et al. [1986] developed their optimisation network for solving linear programming problems. Smith et al. [1989] discussed some practical design problems of the Tank and Hopfield (TH) network along with its stability properties. Kennedy et al. [1988] extended the results of TH to more general non-linear programming problems. Further, they showed that the network introduced by TH was a special case of the canonical non-linear programming network proposed Chua et al. [1984] with capacitors added to account for the dynamic behaviour of the circuit. Gee [1993] overcame some of the limitations of the TH network. In parallel to the development of Hopfield networks to solve TSP we can list the deformable template models, the Elastic Nets (see Durbin et al. [1987]), and the application of Self Organizing Map (see Fort [1988], Kohonen [1990]). Lillo et al. [1993a] analysed the dynamics of the canonical nonlinear programming circuit and showed that it was a gradient system minimising an unconstrained energy function that can be viewed as a penalty method approximation of the original problem. Thus, all of these methods (Hopfield-type networks) use the penalty function method (see Section (10.6.2)) whereby a constrained optimisation problem is approximated by an unconstrained optimisation problem. This is due to the network property of reducing their energy

Electronic copy available at: https://ssrn.com/abstract=3307566

function during evolution, leading to a local or global minimum. Some authors proposed an exact penalty function approach and presented a neural optimisation network for solving constrained optimisation problems (see Lillo et al. [1993b], Mladenov et al. [1999]). The parallel nature of some neural network models seems to be very promising in reducing computation time (see Cichocki et al. [1993], Takefuji et al. [1996]). However, the problem of local minima is the main limitation of the approach and several techniques have been proposed to overcome it, such as stochastic networks, simulated annealing, and other global optimisation methods. Villarrubia et al. [2017] proposed to apply a multilayer perceptron to approximate the objective functions. The same process could be followed in the restrictions. The objective function is approximated with a non-linear regression with the objective of obtaining a new function that facilitates the solution of the optimisation problem. The activation function of the neural network must be selected so that the derivative of the transformed objective functions should be polynomial.

12.2.1.2 Mapping optimisation problems to networks

Since a Hopfield network (see Section (4.1.1)) performs as a minimiser of its energy function, if the optimisation problem can be coded as an energy function, then a network that corresponds to this energy function can be used to minimise (locally or globally) the function and thus provide an optimal or near-optimal solution. Thus, the procedure starts with the construction of the energy function and then the parameters of the network (number of units, weights of connection, thresholds, update policy or even the dynamics) are adjusted to reflect the problem. Then the network is initialised to some initial state and is let to run until it comes to equilibrium from where a solution can be drawn. However, in order to construct the appropriate energy function the latter must be quadratic to satisfy Equation (4.1.3). The most common approach is the penalty (or cost) function approach (see Section (10.6)). The energy function is initialised to the objective function of the problem and for each constraint a penalty term is added. Thus, the problem from a constrained optimisation form is reduced to an unconstrained minimisation problem.
As soon as the energy function has been constructed, the elements of the network (number of units, weights, thresholds) can be derived. For each unit i the energy difference ΔE_i is calculated (derivatives can be used for this purpose) and it is transformed into a form similar to Equation (4.1.4). By analogy, the coefficient of v_j will give the weight w_{ij}, whereas the constant terms will give the threshold θ_i. Note, as the problem size grows, the network size (units and connections) can be intractable large and inapplicable to practical domains. The limitations are due to storage and computing power requirements, but several techniques for parallel implementations have been proposed. The initialisation of the network state is a problem since if it is initialised to a state corresponding to a possible solution, it will be stuck as all the constraints are satisfied, although this solution may not be optimal. Such states corresponds to local minima and are not desirable as initial states. Moreover, different initialisation may lead to different solutions, with different quality and/or computation time. One solution is to consider a random state for initialisation.

12.2.2 Some optimisation models

We briefly present some simple networks used for optimisation problems.

12.2.2.1 Discrete synchronous Hopfield network

In the discrete synchronous Hopfield network, the dynamics of the system are given by Equation (4.1.1) and all the units are updated in parallel. The network is not guaranteed to decrease at each step and since the system may fail to come into equilibrium a solution is not always derived. However, it has the advantage of being fully parallelisable.

12.2.2.2 Discrete asynchronous Hopfield network

This model is similar to the previous one with the difference that the units are updated sequentially. The network will eventually come into equilibrium, from where a solution can be drawn. However, the system can be easily trapped into a local minimum and is highly dependent on the initial state. Further, it can not be parallelised since it is strictly sequential.

Electronic copy available at: https://ssrn.com/abstract=3307566

12.2.2.3 Analog Hopfield network

In the analog Hopfield network, the dynamics of the system are given by Equation (4.1.1). Note, synchronous, asynchronous or continuous updating can be applied. Although the output of such networks is continuous they can be used to solve discrete optimisation problems. In the equilibrium state the outputs closer to 1 are taken as 1 and the output closer to 0 (-1) are taken as 0 (-1). When using the sigmoid function, if we start the network at the temperature T where we want to measure the outputs, it may take a long time to come to equilibrium. In general, the simulated annealing technique is applied. We start the network at a relatively high temperature and gradually we cool it down. The potential disadvantage of this network is that it may be trapped in local minima inside the hypercube without reaching any of the corners.

12.2.2.4 Boltzmann machine

The operation of the Boltzmann machine integrates the dynamics of the discrete asynchronous Hopfield model with the simulated annealing technique. At each step t a unit i of the network is selected randomly and the energy difference ΔE_i that will be caused by a change of its state is calculated using Equation (4.1.4). If ΔE_i is negative (the energy is decreased) the change is definitely accepted, otherwise it is accepted with probability $P_{B,i}(t)$ that depends on the quantity $e^{\frac{\Delta E_i}{T_B}}$ where T_B is a temperature that decreases according to some annealing schedule. In general, the Metropolis criterion is used as the acceptance criterion, which is given by

$$P_{B,i}(t) = \begin{cases} 1 \text{ if } \Delta E_i(t) < 0 \\ \frac{1}{1+e^{\frac{\Delta E_i}{T_B}}} \text{ if } \Delta E_i(t) \geqslant 0 \end{cases}$$

The logarithmic schedule can be used to decrease the temperature:

$$T_B(t) = \frac{T_B(t-1)}{1 + t\log(1+r)}$$

where r is a parameter that adjusts the speed of the schedule.
The Boltzmann machine can be very effective when used with the appropriate annealling schedule and is able to perform a wide exploration of the problem state space. However, it is strictly sequential and can not be parallelised. Several attempts at parallelising its operation (group updates) were proposed, all having to cope with some kind of trade-off between the solution quality and the actual speed-up.

12.2.2.5 Cauchy machine

The Cauchy machine extends the discrete synchronous Hopfield model. The behaviour of a unit i at each time step t is given by

$$v_i(t) = \Theta(u_i(t))$$

where u_i is the activation in Equation (4.1.2). During the operation, the activation is updated using the following motion equation (gradient descent dynamics)

$$\frac{du_i(t)}{dt} = -\frac{\partial E(v)}{\partial v_i}$$

For simulation purposes, we use the first-order approximation for the above dynamics, getting

$$\frac{\Delta u_i}{\Delta t} = -\frac{\Delta E_i}{\Delta v_i} \text{ and } u_i(t+\Delta t) = u_i(t) + \Delta u_i$$

Note, the activation plays the role of an accumulator, a sort of a memory that stores the cumulative activation of the unit during the network's operation time. Hence, the probability of two units to change state simultaneously is

Electronic copy available at: https://ssrn.com/abstract=3307566

reduced significantly, so that oscillation phenomena are avoided. Consequently, the system will eventually come to equilibrium. This is strengthened also by the fact that each unit follows the above gradient descent dynamics. When the energy function is given by Equation (4.1.3), then using Equation (4.1.4) and the above gradient descent dynamics we can derive the motion equation for each unit i as

$$\frac{\Delta u_i}{\Delta t} = \sum_{j=1}^{n} w_{ij}v_j + \theta_i \text{ and } u_i(t+\Delta t) = u_i(t) + \big(\sum_{j=1}^{n} w_{ij}v_j + \theta_i\big)\Delta t$$

A state vector v constitutes an equilibrium state for the network if for all i the following condition is satisfied

$$\big(v_i = 1 \text{ and } \Delta u_i \geqslant 0\big) \text{ or } \big(v_i = 0 \text{ and } \Delta u_i \leqslant 0\big)$$

In order to provide the system with stochastic hill-climbing capabilities, the Distributed Cauchy Machine has been developed, where Cauchy color noise is added in the updating procedure. The output of unit i at each time step t is stochastically updated with probability $P_{C,i}(t)$ which depends on the Cauchy distribution:

$$P_{C,i}(t) = \begin{cases} s_i(t) \text{ if } v_i(t) = 0 \\ 1 - s_i(t) \text{ if } v_i(t) = 1 \end{cases}$$

where

$$s_i(t) = P(v_i(t) = 1) = \frac{1}{2} + \frac{1}{\pi} arctan\big(\frac{u_i(t)}{T_C(t)}\big)$$

The virtual temperature $T_C(t)$ is usually given by a fast annealing schedule:

$$T_C(t) = \frac{T_{C,0}}{1+\beta t}$$

where $T_{C,0}$ is the initial temperature and β is real parameter in the range $[0,1]$ controlling the speed of the schedule. In the extreme case where $T_C = 0$ the network becomes deterministic.
In general, the Cauchy machine provides solutions of lower quality than the Boltzmann machine, but it has the advantage of being fully parallelisable. Further, since the activation is cumulative, changes at the state of a unit can be done only after many time steps, for the activation must change sign. The lack of this flexibility becomes more obvious when the system has operated for a long time and the activations have large absolute values.

12.2.2.6 Hybrid Scheme

Papageorgiou et al. [1998] presented the hybrid update scheme as an attempt at combining both the advantages of the Boltzmann and the Cauchy machine. It extends the synchronous discrete Hopfield model and the stochastic update rule is based on a convex combination of the Boltzmann and Cauchy machine update rules. At each time step t, the probability of accepting a state change concerning unit i is given by

$$P_{H,i}(t) = \alpha P_{C,i}(t) + (1-\alpha)P_{B,i}(t)$$

where α is a control parameter in the range $[0,1]$. A large value of α will result to a typical Cauchy machine, whereas a small value will result to a synchronous Boltzmann machine destroying the convergence property. The two temperatures need not be equal and a good choice would be to update $T_C(t)$ according to some annealing schedule and then take $T_B(t) = \lambda T_C(t)$ where λ is an adjustable parameter. Even though the above stochastic update rule accepts changes suggested by the energy difference (through $P_{B,i}(t)$), such changes should be reflected on the activation $u_i(t)$, otherwise at the next time step the unit would return to the previous value, since it is still suggested by the activation. The following rule ensures that this will not happen:

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} \text{if} \quad & \left(u_i(t+\Delta t) <> u_i(t)\right) \text{ and } \left(P_{C,i}(t) < 0.25\right) \text{ and } \left(P_{B,i}(t) > 0.75\right) \\ \text{then} \quad & \left(u_i(t+\Delta t) = -u_i(t)\right) \end{aligned}$$

Lagoudakis [1997] showed that $\left(P_{B,i}(t) > 0.75\right)$ was necessary for the network to converge. Experimental results showed that the solutions produced by the hybrid scheme were similar to the solutions produced by the Boltzmann machine but with larger convergence time, similar to that of the Cauchy machine. However, the hybrid scheme is fully parallelisable, so that the parallel implementation provides solutions similar to that of the Boltzmann machine but in substantially less time.

12.2.3 Approximation with non-linear regression

12.2.3.1 The method

The main idea is to approximate the objective function f with some heuristics methods. Approximation functions are usually defined around a point, which would make it possible to use polynomials to approximate functions by applying the Taylor theorem. Based on this idea, one can solve non-linear optimisation problems by applying Taylor non-linear functions. For instance, Nanculef et al. [2014] applied this method in Frank-Wolfe algorithms, allowing for the linearisation of the objective functions by applying derivatives in a point to calculate the straight line, plane or hyperplane crosses through that point. The solutions are calculated iteratively with a new hyperplane for each iteration. Note, the MAP (Method of Approximation Programming) is a generalisation of the Frank-Wolfe algorithm, which permits to linearise the restrictions.
Villarrubia et al. [2017] proposed to generalise this approach by using the universal approximation theorem (see Theorem (4.1.1) and approximate the objective function with neural networks. That is, rather than calculating a new approximation for each tentative solution, they chose to infer the approximation by applying neural networks. Given a known objective function f, the system generates a dataset in the domain of the variables to train a neural network. The objective function of the optimisation problem is redefined with the multilayer perceptron that transforms the function, making it possible to generate a polynomial equation to resolve the optimisation problem. Finally, when the new objective function is calculated another solution (such as the Lagrangian method) can be applied to resolve the problem.
To define a neural network, it is necessary to establish parameters, such as the connections, number of layers, activation functions, propagation rules etc. In the case of the multilayer perceptron, the authors considered two different stages: the learning stage, and the prediction process. In both stages, the number of layers and activation functions have to be the same.

12.2.3.2 Defining the network

Following the Hopfield networks (see Section (4.1.1)) and multilayer networks (see Section (4.2)), the propagation rule is the weighted sum (see activation in Equation (4.1.2)) defined by

$$u_j(t) = \sum_{i=1}^{n} w_{ij} x_i(t) + \theta_j$$

where w_{ij} is the weight connecting neuron i in the input layer with neuron j in the hidden layer , x_i is the output from neuron i in the input layer, n is the number of neurons in the input layers, t is the pattern, and θ_j is the bias.
We now need to select the activation function Φ of the network such that that the derivatives of the transformed objective functions are polynomial. This is because it will simplify calculations when using the Lagrangian method to solve the optimisation problem. If the function Φ is linear, the output of neuron j is a linear combination of the neurons in the input layer and, consequently, y_j is a linear function. Therefore, if the activation function is the identity,

Electronic copy available at: https://ssrn.com/abstract=3307566

the net output would correspond to the output of the neuron. Thus, if neuron j in the hidden layer has the activation function Φ , the output is given by

$$y_j(t) = \Phi(u_j(t))$$

which is equivalent to a non-linear regression.
Since the multilayer perceptron has three layers, we apply the propagation rule twice in order to transmit the value from the input layer to the neurons in the output layer. If neuron k in the output layer has the activation function Φ_{out}, the output is given by

$$y_k(t) = \Phi_{out}\big(\sum_{j=1}^{m} w_{jk} y_j(t) + \theta_k\big)$$

where m is the number of neurons in the hidden layer. In the special case where Φ_{out} is the identity function, the output simplifies to

$$y_k(t) = \sum_{j=1}^{m} w_{jk} y_j(t) + \theta_k$$

Replacing $y_j(t)$ with its value defined above, the output in neuron k is defined as

$$y_k(t) = \sum_{j=1}^{m} w_{jk} \Phi(u_j(t)) + \theta_k \tag{12.2.1}$$

Using the universal approximation theorem (see Theorem (4.1.1)), we can deduce that there exists $\epsilon > 0$ such that

$$|y_k(t) - f| < \epsilon$$

In the special case where the function Φ is the identity, the output in neuron k from the output layer is calculated as a linear combination of the inputs, so that the function is linear. Thus, if we train the multilayer perceptron with an identity activation function, it would be possible to make an approximation of the trained function. In the case where an optimisation problem has a non-linear objective function, it would then be possible to redefine the function according to Equation (12.2.1) so that it would be linear. However, with this activation function we can not prevent the approximation functions from becoming hyperplane, so that the activation function Φ can not be linear. Another choice would be to consider the arctan activation function because the equation of the derivative [1] is simpler than the expression of the sigmoidal function.

12.2.3.3 The Lagrangian method

In order to solve the optimisation problem defined in Equation (10.1.2), we can approximate the objective function f with the neural network defined in Equation (12.2.1), getting

$$f(x) \approx \sum_{j=1}^{m} w_{jk} \Phi(u_j(t)) + \theta_k \text{ such that } g_i(x) \leqslant 0 \ , i = 1, ..., m$$

We can then use the Lagrangian approach (see Section (10.4)) to solve this optimisation problem. In that case the Lagrangian in Equation (10.4.3) becomes

$$L(x, \lambda) = \sum_{j=1}^{m} w_{jk} \Phi(u_j(t)) + \theta_k + \sum_{i=1}^{m} \lambda_i \cdot g_i(x)$$

[1] $arctan(f) = \frac{f'}{1+f^2}$

Electronic copy available at: https://ssrn.com/abstract=3307566

with the Karush-Kuhn-Tucker (KKT) conditions

$$\begin{aligned}\frac{\partial L}{\partial x_i} &= 0\,,i=1,...,n\\ \frac{\partial L}{\partial \lambda_j} &= 0\,,j=1,...,m\end{aligned}$$

The stationarity condition can be written as

$$\forall j\ \frac{\partial f(x)}{\partial x_j}\pm\sum_{i=1}^{m}\lambda_i\frac{\partial g_i(x)}{\partial x_j}\leqslant 0$$

and the complementary condition is

$$\forall i:\lambda_i g_i(x)=0$$

We clearly see that we need to use an activation function Φ which is analytically differentiable to simplify the derivatives in order to calculate the solution more easily. For restrictions with inequality, it is possible to introduce a threshold based on the training carried out in the neural network. For the threshold to be lower, it would be best to have previously trained the neural network with values in the variables around their definition. That is, in a problem with restrictions given by

$$\begin{aligned}&a_i\leqslant x_i\leqslant b_i\\ &x_i\geqslant a_i\\ &x_i\leqslant b_i\end{aligned}$$

we should generate a dataset for the training phase that matches the restrictions for the variable x_i. The lower the difference among consecutive values, the lower the error to define the threshold. The authors got the restriction

$$g_s(x)\leqslant 0$$

and given Equation (12.2.1), they got

$$\sum_{j=1}^{m}w_{jk}\Phi(u_j(t))+\theta_k+\mu_s=0$$

Finally, the optimisation problem becomes

$$f(x)=\sum_{j=1}^{m^1}w_{jk}^1\Phi\big(\sum_{i=1}^{n^1}w_{ij}^1x_i^1(t)+\theta_j^1\big)+\theta_k^1$$

such that

$$\begin{aligned}\sum_{j=1}^{m^1}w_{jk}^1\Phi\big(\sum_{i=1}^{n^1}w_{ij}^1x_i^1(t)+\theta_j^1\big)+\theta_k^1+\mu_s^1 &= 0\\ &\vdots\\ \sum_{j=1}^{m^n}w_{jk}^n\Phi\big(\sum_{i=1}^{n^n}w_{ij}^nx_i^n(t)+\theta_j^n\big)+\theta_k^n+\mu_s^n &= 0\end{aligned}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

12.3 Solving combinatorial optimisation problems

We have presented continuous optimisation problems in Chapter (10). We are now going to briefly discuss discrete optimisation problems, which consists in finding an optimal object from a set of objects. In an optimisation problem with discrete variables (discrete optimisation), we are looking for an object such as an integer, permutation or graph from a finite (or possibly countably infinite) set. Among the different branches of discrete optimisation, combinatorial optimisation refers to problems on graphs, matroids and other discrete structures. A graph is a structure amounting to a set of objects in which some pairs of the objects are in some sense related. The objects correspond to mathematical abstractions called vertices (also called nodes or points) and each of the related pairs of vertices is called an edge (also called an arc or line). Typically, a graph is depicted in diagrammatic form as a set of dots for the vertices, joined by lines or curves for the edges. Note, some some combinatorial optimisation problems can be solved with integer programs and vice versa. The domain of those optimisation problems has discrete set of feasible solutions such that exhaustive search is not tractable and one is left to finding the best solution. Some examples of combinatorial optimisation problems are shortest paths and shortest path trees, flows and circulations, spanning trees, matching and matroid problems. In graph theory, the shortest path problem consists in finding a path between two vertices in a graph such that the sum of the weights of its constituent edges is minimised. In this section, we will focus on the travelling salesman problem (TSP), which consists in finding the shortest path going through every vertex exactly once, and returning to the start.

12.3.1 Defining the problem

12.3.1.1 Description

Some common problems involving combinatorial optimisation are the travelling salesman problem (TSP) and the minimum spanning tree problem (MST). The travelling purchaser problem and the vehicle routing problem are both generalisations of TSP. The TSP asks the following question: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city? Thus, given a graph, one needs to search the space of permutations to find an optimal sequence of nodes with minimal total edge weights (tour length). In practice, TSP solvers rely on handcrafted heuristics that guide their search procedures to find competitive (and even optimal) tours efficiently. Those heuristics describe how to navigate the space of feasible solutions in an efficient manner (see Applegate et al. [2006] [2011]). More generic solvers, rely on a combination of local search algorithms and metaheuristics (see Google 2016). However, if the problem statement slightly changes, the heuristics need to be revised.

12.3.1.2 Formalising the problem

A combinatorial optimisation problem A is a quadruple (I, f, m, g) where

- I is a set of instances.
- given an instance $x \in I$, $f(x)$ is the set of feasible solutions.
- given an instance x and a feasible solution y of x, $m(x, y)$ denotes the measure of y which is usually a positive real.
- g is the goal function, and is either $\min$ or $\max$.

The goal is then to find, for some instance x, an optimal solution, that is, a feasible solution y with

$$m(x, y) = g\big(m(x, y^{'})|y^{'} \in f(x)\big)$$

For each combinatorial optimisation problem, there is a corresponding decision problem that asks whether there is a feasible solution for some particular measure m_0.

Electronic copy available at: https://ssrn.com/abstract=3307566

In the field of approximation algorithms, algorithms are designed to find near-optimal solutions to hard problems. The usual decision version is then an inadequate definition of the problem since it only specifies acceptable solutions. Nonetheless, even though we could introduce suitable decision problems, the problem is more naturally characterised as an optimisation problem.
An algorithm is said to be of polynomial time if its running time is upper bounded by a polynomial expression in the size of the input for the algorithm, that is, $T(n) = O(n^k)$ for some positive constant k. Problems for which a deterministic polynomial time algorithm exists belong to the complexity class P, which is central in the field of computational complexity theory.
An NP-optimisation problem (NPO) is a combinatorial optimisation problem with the following additional conditions:

- the size of every feasible solution $y \in f(x)$ is polynomially bounded in the size of the given instance x
- the languages $\{x|x \in I\}$ and $\{(x, y)|y \in f(x)\}$ can be recognised in polynomial time
- m is polynomial-time computable

This implies that the corresponding decision problem is in NP. Here, the polynomials are functions of the size of the respective functions inputs, not the size of some implicit set of input instances. A problem is additionally called a P-optimisation (PO) problem, if there exists an algorithm which finds optimal solutions in polynomial time. Often, when dealing with the class NPO, one is interested in optimisation problems for which the decision versions are NP-complete.

12.3.2 Solving the TSP

12.3.2.1 A review

Motivated by the recent advancements in sequence-to-sequence (seq2seq) learning (see Sutskever et al. [2014]), neural networks are again the subject of study for optimisation in various domains (see Chen et al. [2016]), including discrete ones (see Zoph et al. [2016]). In particular, Vinyals et al. [2015b] revisited the travelling salesman problem (TSP) with the Pointer Networks, where a recurrent network with non-parametric softmaxes is trained in a supervised manner to predict the sequence of visited cities.
As discussed in Section (8.1), supervised learning is about learning a mapping from training inputs to outputs. Thus, in order to learn an optimisation problem we would need to have access to optimal labels, which is in general not the case. Bello et al, [2017] followed the reinforcement learning (RL) paradigm to tackle combinatorial optimisation, leading to Neural Combinatorial Optimisation. The method allows one to compare the quality of a set of solutions using a verifier (control), and provide some reward feedbacks to a learning algorithm. They considered two approaches based on policy gradients (see Williams et al. [1992]): (1) RL pretraining, uses a training set to optimise a recurrent neural network (RNN) that parameterises a stochastic policy over solutions, using the expected reward as objective. At test time, the policy is fixed, and one performs inference by greedy decoding or sampling. (2) active search, involves no pretraining. It starts from a random policy and iteratively optimises the RNN parameters on a single test instance, again using the expected reward objective, while keeping track of the best solution sampled during the search. The authors found that combining RL pretraining and active search works best in practice. They compared their approach against the supervised learning approach to the TSP proposed by Vinyals et al. [2015b]. The latter consider Pointer Networks which is a neural architecture to learn the conditional probability of an output sequence with elements that are discrete tokens corresponding to positions in an input sequence.

12.3.2.2 The problem

For simplicity of exposition we are going to present a 2D Euclidean travelling salesman problem (TSP). Given an input graph, represented as a sequence of n cities in a two dimensional space $s = \{x_i; i = 1, ..., n\}$, where each $x_i \in \mathbb{R}^2$, our objective is to find a permutation of the points π (called a tour) that visits each city once and has the minimum total length. The length of a tour, defined by a permutation π, is given by

Electronic copy available at: https://ssrn.com/abstract=3307566

$$L(\pi|s) = \|x_{\pi(n)} - x_{\pi(1)}\|_2 + \sum_{i=1}^{n-1} \|x_{\pi(i)} - x_{\pi(i+1)}\|_2$$

where $\|\cdot\|_2$ is the L_2 norm. We want to learn the parameters of a stochastic policy $p(\pi|s)$ [2], which given an input set of points s, assigns high probabilities to short tours and low probabilities to long tours.
Sutskever et al. [2014] proposed a neural network architecture that uses the chain rule to factorise the probability of a tour as

$$p(\pi|s) = \prod_{i=1}^{n} p\big(\pi(i)|\, \pi(< i), s\big)$$

and then uses individual softmax modules to represent each term on the RHS of the above equation. One could then use a sequence to sequence model (seq2seq) to address the TSP where the output vocabulary is $\{1, 2, ..., n\}$. However, such trained network can not generalise to input with more than n cities and we need access to the true output permutation to train the model.
Vinyals et al. [2015b] proposed the Pointer Network that generalises the approach beyond a pre-specified graph size by making use of a set of non-parametric softmax modules, similar to the attention mechanism from Bahdanau et al. [2015]. It allows the model to effectively point to a specific position in the input sequence rather than predicting an index value from a fixed-size vocabulary.
The authors proposed training a pointer network using a supervised loss function comprising conditional log-likelihood, which factors into a cross entropy objective between the network's output probabilities and the targets provided by a TSP solver. However, as pointed out by Bello et al. [2017], the performance of the model is tied to the quality of the supervised labels, getting high-quality labelled data is expensive and may be infeasible for new problem statements. At last, it will be equivalent to replicating the results of another algorithm (TSP solver).

12.3.3 The REINFORCE algorithm

As discussed in Section (9.4.3), the REINFORCE algorithm is a policy gradient method that uses the likelihood ratio trick. Basically it increases the probability that a policy will make a rewarding action given a state s. It is a family of reinforcement learning methods which directly update the policy weights θ through the following rule:

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) v_t \tag{12.3.2}$$

where α is the learning rate, $\pi_\theta(\cdot|s_t)$ is the policy (mapping actions to probabilities) (see Definition (9.3.1)), and v_t is a sample of the value function at time t collected from experience.
It iteratively updates the agent's parameters by computing the policy gradient as follows:

- Update the parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return v_t as an unbiased sample of $Q_{\pi_\theta}(s_t, a_t)$ solve Equation (12.3.2)

Previously to the REINFORCE algorithm, one had to learn a value function through function approximation and then derive a corresponding policy (see Section (9.4)), but often learning a value function can be intractable. This method provides a way to directly optimise policies to get around this problem. See details in Algorithm (36).

[2] The authors use a different notation than ours. It corresponds to the parametric policy $\pi_\theta(s, a)$.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 36 REINFORCE Algorithm

Require: : training set S, number of training steps T
1: Initialise θ
2: **for** each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
3: **for** $t = 1$ to $T - 1$ **do**
4: $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) v_t$
5: **end for**
6: **end for**
7: **Return** θ

12.3.4 Applying reinforcement learning

In the case of combinatorial optimisation, Bello et al. [2017] considered reinforcement learning (RL) because these problems have simple reward mechanism that can even be used at test time. They considered two approaches based on policy gradients (see Section (9.4.3)) where we use neural networks to learn the policy. They used the pointer network architecture as the policy model to parametrise the policy $p(\pi|s)$. Their pointer network comprises two recurrent neural network (RNN) modules, encoder and decoder, both of which consist of Long Short-Term Memory (LSTM) cells (see Section (5.2)).

1. The encoder network reads the input sequence s one city at a time, and transforms it into a sequence of latent memory states $\{enc_i\}_{i=1}^n$ where $enc_i \in \mathbb{R}^2$. The input to the encoder network at time step i is a d-dimensional embedding of a 2D point x_i, which is obtained via a linear transformation of x_i shared across all input steps.

2. The decoder network also maintains its latent memory states $\{dec_i\}_{i=1}^n$ where $dec_i \in \mathbb{R}^2$. At each step i, it uses a pointing mechanism to produce a distribution over the next city to visit in the tour. Once the next city is selected, it is passed as the input to the next decoder step. The input of the first decoder step (denoted $< g >$) is a d-dimensional vector treated as a trainable parameter of the neural network.

The attention function takes as input a query vector $q = dec_i \in \mathbb{R}^d$ and a set of reference vectors $ref = \{enc_1, ..., enc_k\}$ where $enc_i \in \mathbb{R}^d$, and predicts a distribution $A(ref, q)$ over the set of k references. This probability distribution represents the degree to which the model is pointing to reference r_i upon seeing query q.
The authors used model-free policy-based Reinforcement Learning to optimise the parameters of the pointer network denoted θ. The training objective is the expected tour length which, given an input graph s, is defined as

$$J(\theta|s) = E_{\pi \sim p_\theta(\cdot|s)}[L(\pi|s)]$$

where the length of a tour $L(\pi|s)$ acts as the reward.
During training, the graphs are drawn from a distribution $\mathcal{S}$ and the total training objective involves sampling from the distribution of graphs

$$J(\theta) = E_{s \sim S}[J(\theta|s)]$$

The parameters are optimised with the policy gradient methods (see Section (9.4.3)) and the stochastic gradient descent (see Section (10.2)). The gradient of the above objective is formulated with the REINFORCE algorithm (see Section (9.4.3.2)) as follows

$$\nabla_\theta J(\theta|s) = E_{\pi \sim p_\theta(\cdot|s)}\big[\big(L(\pi|s) - b(s)\big)\nabla_\theta \log p_\theta(\pi|s)\big]$$

where $b(s)$ denotes a baseline function, independent from π, that estimates the expected tour length to reduce the variance of the gradients. Drawing B i.i.d. sample graphs $s_1, s_2, ..., s_B \sim \mathcal{S}$ and sampling a single tour per graph, $\pi_i \sim p_\theta(\cdot|s_i)$, the above gradient is approximated with with Monte Carlo sampling as

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\nabla_\theta J(\theta) \approx \frac{1}{B}\sum_{i=1}^{B}\Big(L(\pi_i|s_i) - b(s_i)\Big)\nabla_\theta \log p_\theta(\pi_i|s_i)$$

In general it is taken to be an exponential moving average of the rewards obtained by the network over time to account for the fact that the policy improves with training. However, we can not differentiate between the different input graphs. Since a parametric baseline to estimate the expected tour length improves the learning, the authors introduced an auxiliary network, called a critic, and parametrised it by θ_v to learn the expected tour length (see Section (9.4.4)). It is trained with stochastic gradient descent on a mean squared error objective between its prediction $b_{\theta_v}(s)$ and the actual tour lengths sampled by the most recent policy. The objective is given by

$$\mathcal{L}(\theta_v) = \frac{1}{B}\sum_{i=1}^{B}\|b_{\theta_v}(s_i) - L(\pi_i|s_i)\|_2^2$$

where B is the batch size. The critic to map a sequence s into a baseline $b_{\theta_v}(s)$ is made of three neural network modules

1. an LSTM encoder
2. an LSTM process block
3. a 2-layer ReLU neural network decoder

The training algorithm, closely related to the asynchronous advantage actor-critic (A3C) (see Mnih et al. [2016]), is given in Algorithm (37).

Algorithm 37 Actor-critic Training Algorithm

Require: : training set S, number of training steps T, batch size B

Initialise pointer network parameters θ
Initialise critic network parameters θ_v
for $t = 1$ to T **do**
  $s_i \sim SampleInput(S)$ for $i \in \{1, ..., B\}$
  $\pi_i \sim SampleSolution(p_\theta(\cdot|s_i))$ for $i \in \{1, ..., B\}$
  $b_i \leftarrow b_{\theta_v}(s_i)$ for $i \in \{1, ..., B\}$
  $g_\theta \leftarrow \frac{1}{B}\sum_{i=1}^{B}\big(L(\pi|s) - b(s)\big)\nabla_\theta \log p_\theta(\pi|s)$
  $\mathcal{L}_v \leftarrow \frac{1}{B}\sum_{i=1}^{B}\|b_i - L(\pi_i)\|_2^2$
  $\theta \leftarrow ADAM(\theta, g_\theta)$
  $\theta_v \leftarrow ADAM(\theta_v, \nabla_\theta \mathcal{L}_v)$
end for
Return θ

Since the evaluation of a tour length is inexpensive, one can follow classical search by considering multiple candidate solutions per graph and selecting the best one. The authors considered two search strategies

1. Sampling: sample multiple candidate tours from the stochastic policy $p_\theta(\cdot|s)$ and selecting the shortest one.
2. Active search (on untrained model): the idea is to reward information obtained from the sampled solutions by refining the parameters of the stochastic policy p_θ during inference to minimise $J(\theta|s)$ on a single test input s. It applies policy gradients as in Algorithm (37), but it draws Monte Carlo samples over candidate solutions $\pi_1, ..., \pi_B \sim p_\theta(\cdot|s)$ for a single test input. It also consider an exponential moving average baseline. Note, it is independent of distribution.

Electronic copy available at: https://ssrn.com/abstract=3307566

When it comes to solving other optimisation problems, one should use Pointer Network when the output is a permutation or a subset of the input, and the seq2seq model for other kinds of structured outputs. When feasibility is known at decoding time one simply assign a zero probability to a non-feasible solution (branches). However, for many combinatorial problems finding feasible solutions is a challenge as one need to search subtrees of branches. Rather than explicitly constraining the model to only sample feasible solutions, one can let the model learn the constraints by augmenting the objective function with a penalising term, similarly to penalty methods in constrained optimisation (see Section (10.6)). However it does not guarantee to always sample feasible solutions at inference time.

Electronic copy available at: https://ssrn.com/abstract=3307566

Part V

Applications

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 13

Mathematical Finance

Since financial markets exhibit non-Gaussian response time distribution leading to more skewness and kurtosis than it is consistent with the Geometric Brownian Motion model of Black-Scholes, neural networks (NNs) are good candidates to learn these highly nonlinear relationships. We saw in Chapter (5) that recurrent neural networks could be used to predict the stock market. Further, we saw in Chapter (8) that supervised learning was concerned with solving function estimation problems. These methods can be used in finance to calibrate option prices in view of interpolating / extrapolating the volatility surface. It can also be used in risk management to fit options prices at the portfolio level in view of performing some credit risk analysis. Further, we saw in Chapter (9) that reinforcement learning (RL) consisted in solving problems involving an agent interacting with an environment, which provides numeric reward signals. The goal being to learn how to take actions in order to maximise reward. As such it can be used for selecting optimal portfolios. We will see in this chapter that it can also be used to price European and American options when the principle of hedging first and pricing second is applied. Contrary to classical option pricing theory which is rooted in stochastic calculus and continuous-time models, machine learning is rooted in dynamic programming (DP) and reinforcement learning. However, solving financial problems in that setting requires an exhaustive amount of data, which is in general not there. As such, one must rely on some models to generate the missing data.

13.1 Option pricing

13.1.1 The framework

A derivative, or contingent claim, is a financial instrument whose value depends on the values of other more basic underlying variables being the prices of traded securities. An option is a security giving the right to buy or sell an asset, subject to certain conditions, within a specified period of time. The holder of an option has the right to exercise the option but it is not an obligation. European, or vanilla, options such as a call option give the holder the right to buy the underlying asset by a certain date called the maturity for a specified price called the exercise price or strike price, while the put option gives him the right to sell it. We let K be the strike price and S_T be the underlying price at time T, then the payoff from a long position in a European call option is

$$\max\big(S_T - K, 0\big) = (S_T - K)^+$$

so that the option will be exercised if $S_T > K$.
We display in Figure (13.1) the call price and its linear approximation versus different strike levels.
As a result of the concept of absence of arbitrage opportunities (AAO) (see Harrison and Kreps [1979]), contingent claims can be valued by taking expectation of their discounted payoffs under the risk-neutral measure. Therefore, the price of a European contingent claim $C(t, x)$ on $]0, T] \times]0, +\infty[$ under the risk-neutral measure $\mathbb{Q}$ is

Electronic copy available at: https://ssrn.com/abstract=3307566

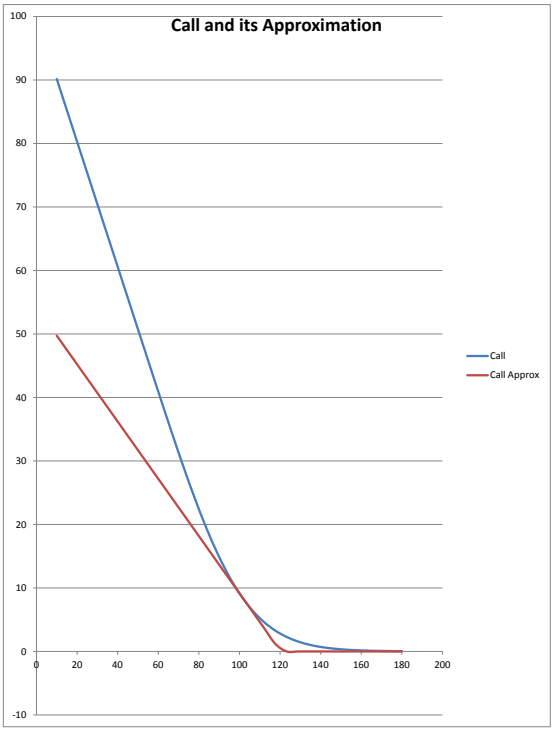


Figure 13.1: The call price and its linear approximation vs strikes.

$$C(t, X_t) = E^Q[e^{-\int_t^T r_s ds} h(X_T)|\mathcal{F}_t] \tag{13.1.1}$$

where X_T is a $\mathbb{Q}$-random variable and h is a sufficiently smooth payoff function.

13.1.1.1 Defining the replicating portfolio

Initial work on the valuation of options has been expressed in terms of warrants (see Sprenkle [1961], Samuelson [1965] and Samuelson et al. [1969]), all providing valuation formulas involving one or more arbitrary parameters. Thorp et al. [1967] obtained an empirical valuation formula for warrants which they used to calculate the ratio of shares of stock to options needed to create a hedge position by going long in one security and short in the other. However, they failed to see that in equilibrium, the expected return on such a hedge position must be equal to the return on a riskless asset. Using this equilibrium condition, Black et al. [1973] derived a theoretical valuation formula. To do so, they assumed the following ideal conditions in the market

1. The short-term interest rate is known and constant through time.
2. The stock price follows a random walk in continuous time with a variance rate proportional to the square of the stock price.

Electronic copy available at: https://ssrn.com/abstract=3307566

3. The stock pays no dividends or other distributions.
4. The option is European.
5. There are no transaction costs in buying or selling the stock or the option.
6. It is possible to borrow any fraction of the price of a security to buy it or to hold it, at the short-term interest rate.
7. There are no penalties to short selling.

Under these assumptions, the value of the option will only depend on the price of the stock and time, so that it is possible to create a hedged position, consisting of a long position in the stock and a short position in the option. In a complete market, every terminal payoff can be replicated by a portfolio, so that the problem of defining the price of an option is linked with that of finding a replicating portfolio. That is, using the option premium, one can buy and sell the underlying to hedge risk against the derivative. Following that argument, Black and Scholes defined the price of a derivative option as the price of its hedge. It lead to the formulation of the local no-arbitrage condition:

Definition 13.1.1 *The local no-arbitrage condition*
The local no-arbitrage condition states that in a one-factor case, a portfolio can be constructed from any two securities and continually rebalanced at each time, in such a way that it is instantaneously hedged, in the sense that its instantaneous return is certain and hence must equal the short-term, spot interest rate.

Thus, knowing the dynamics of the underlying price we need to formalise the evolution of the self-financing replicating portfolio dynamically managed (without adding or removing capital during the trading period). Following the approach in Section (13.3.1.1), we put ourself in the Black-Scholes world and we assume a market made of a risk-free asset S_t^0 and a risky asset with price S_t. We model the dynamics of the future stock prices under the historical measure $\mathbb{P}$ as

$$\begin{aligned} dS_t &= S_t\big(\mu_t dt + \sigma_t d\widehat{W}_t\big) \\ \mu_t &= r_t + \sigma_t \lambda_t \end{aligned}$$

We denote the quantity of cash that one can hold as $\beta(t)S_t^0$ with dynamics given by

$$dS_t^0 = S_t^0 r dt$$

We build a self-financing portfolio V made of cash and stocks with value at time t being

$$V_t = \alpha(t)S_t + \beta(t)S_t^0$$

and we denote the value invested in stocks as $\pi_t = \alpha(t)S_t$. Over a very short period of time the variations of the portfolio are only due to the variations of the asset price and the interest rate rewarding the cash held at the bank. Hence, the portfolio dynamics are given by

$$dV_t = \alpha(t)dS_t + \beta(t)dS_t^0$$

Replacing the respective dynamics in the above equation, the self-financing portfolio satisfies the stochastic differential equation (SDE)

$$dV_t = r_t V_t dt + \pi_t \sigma_t (d\widehat{W}_t + \lambda_t dt)$$

Market completeness is equivalent to assuming that markets are efficient, that is, the price of an asset at a given time contains all past information as well as the market expectation of its future value. This means that there exists a strategy leading to a final P/L of null value in all possible future configuration. As a result, we can conclude that

Electronic copy available at: https://ssrn.com/abstract=3307566

the option price is equal to the value of the replicating portfolio at the initial time. A direct application of the above assumption is that we can express the Black-Scholes price dynamics as

$$dC_{BS}(t,S_t) = r_t C_{BS}(t,S_t)dt + \pi_t\sigma_t(d\widehat{W}_t + \lambda_t dt)$$

Using Girsanov's theorem, we can re-express that price dynamics under the risk-neutral measure $\mathbb{Q}$ by applying the change of measure

$$W_t = \widehat{W}_t + \int_0^t \lambda_s ds$$

where W_t is a $\mathbb{Q}$-Brownian motion. Therefore, the Black-Scholes price at all future time is a solution to the following stochastic differential equation with terminal condition

$$\begin{aligned} dC_{BS}(t,S_t) &= r_t C_{BS}(t,S_t)dt + \pi_t\sigma_t dW_t \\ C_{BS}(T,S_T) &= h(S_T) \end{aligned} \tag{13.1.2}$$

where the pair $(\alpha(t),\beta(t))$ allows replication of the contingent claim paying $h(S_T)$ at maturity. Note, a consequence of the option price being expressed under the risk-neutral measure is that it neither depend on the return μ of the risky asset nor the market price of risk λ. However, the risk due to the variations of the risky asset is still present and significantly impact the option price via the volatility parameter σ. Thus, to avoid arbitrage, a market price of risk is computed such that the discounted spot price is a martingale with respect to the equivalent probability measure. In the risk-neutral measure, the drift of the spot price is equal to the risk-free rate so that the key parameter becomes the volatility of the dynamics. Volatility is a key parameter when risk managing option prices as it allows the portfolio manager, or trader, to define his hedging strategies by computing the necessary Greeks. The Vega is a risk measure quantifying exposure to a misspecified volatility.

13.1.1.2 The BS-formula

In the special case where we assume that the stock prices $(S_t^x)_{t\geqslant 0}$ are lognormally distributed and pay a continuous dividend, we can derive most of the contract market values in closed-form solution. Balck-Scholes [1973] formalised a method to infer such a solution by developing a theory of hedging and replication by dynamic strategies. They derived the price of a call option seen at time t with strike K and maturity T as

$$C_{BS}(t,x,K,T) = xe^{-q(T-t)}N(d_1(T-t,F(t,T),K)) - Ke^{-r(T-t)}N(d_2(T-t,F(t,T),K)) \tag{13.1.3}$$

where $F(t,T) = xe^{(r-q)(T-t)}$ is the forward price and $N(\cdot)$ is the normal cumulative distribution function (CDF). Further, we have

$$d_2(t,x,y) = \frac{1}{\sigma\sqrt{t}}\log\frac{x}{y} - \frac{1}{2}\sigma\sqrt{t} \text{ and } d_1(t,x,y) = d_2(t,x,y) + \sigma\sqrt{t}$$

We describe a few Greeks in the Black-Scholes formula that will be used later on to devise our parametric model. The option in the Black-Scholes model is hedged with a portfolio containing

$$\Delta(t,S_t) = \partial_x C_{BS}(t,S_t,K,T) = Re(t,T)N(d_1(T-t,S_te^{(r-q)(T-t)},K)) \tag{13.1.4}$$

stocks, since $\frac{d}{dx}d_i = \frac{1}{x\sigma\sqrt{T-t}}$ for $i=1,2$. Note, $x = S_t$ and $Re(t,T) = e^{-q(T-t)}$. We let $P(t,T)$ be the discount factor, $Re(t,T)$ be the repo factor and we define

$$\eta = \frac{K}{F(t,T)} = \frac{KP(t,T)}{xRe(t,T)} \tag{13.1.5}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

to be the forward moneyness of the option. Expressing the strike in terms of the forward price $K = \eta F(t,T)$, the call price in Equation (13.1.3) becomes

$$C_{TV}(t,x,K,T)\big|_{K=\eta F(t,T)} = xe^{-q(T-t)}\left(N(d_1(\eta,\omega(T-t)) - \eta N(d_2(\eta,\omega(T-t)))\right) \tag{13.1.6}$$

where $d_2(\eta,\omega(t)) = \frac{1}{\sqrt{\omega(t)}}\log\frac{1}{\eta} - \frac{1}{2}\sqrt{\omega(t)}$, or equivalently

$$d_2(\eta,\omega(t)) = -\frac{1}{\sqrt{\omega(t)}}\log\eta - \frac{1}{2}\sqrt{\omega(t)} \text{ and } d_1(\eta,\omega(t)) = d_2(\eta,\omega(t)) + \sqrt{\omega(t)} \tag{13.1.7}$$

which only depends on the forward moneyness and the total variance.

13.1.1.3 The implied volatility

In the BS-model, the volatility is a parameter quantifying the risk associated to the returns of the underlying asset, and it is the only unknown variable. In principle, one should use the future volatility in the BS-formula, but its value is not known and needs to be estimated. As a result, practitioners use the implied volatility (IV) when managing their books, assuming that the IV bears valuable information on the asset price process and its dynamics.
We consider non-negative price processes, and assume a perfectly liquid market for European calls. That is, call option prices for all strikes $K > 0$, and maturities $T \geqslant 0$ are known in the market. We define a price surface as follows (see Roper [2010]):

Definition 13.1.2 *A call price surface parametrised by s is a function*

$$\begin{aligned} C : [0,\infty) \times [0,\infty) &\to \mathbb{R} \\ (K,T) &\to C(K,T) \end{aligned}$$

along with a real number $s > 0$.

However, market practice is to use the implied volatility when calculating the Greeks of European options. Thus, we let the implied volatility (IV) be a mapping from time, spot prices, strike prices and expiry days to $\mathbb{R}^+$

$$\Sigma : (t, S_t, K, T) \to \Sigma(t, S_t; K, T)$$

Hence, we define the implied volatility as follow:

Definition 13.1.3 *Given the option price $C(t,S_t,K,T)$ at time t for a strike K and a maturity T, the market implied volatility $\Sigma(t,S_t;K,T)$ satisfies*

$$C(t,S_t,K,T) = C_{BS}(t,S_t,K,T;\Sigma(K,T)) \tag{13.1.8}$$

where $C_{BS}(t,S_t,K,T;\sigma)$ is the Black-Scholes formula for a call option with volatility σ.

Given (S,K,T) and the price of an option, there is a unique implied volatility associated to that price, since

$$\partial_\sigma C_{BS}(t,S_t,K,T;\sigma) > 0$$

Thus, the implied volatility is obtained by inverting the Black-Scholes formula $C_{BS}^{-1}(C(t,S_t,K,T);K,T)$. Consequently, we refer to the two-dimensional map

$$(K,T) \to \Sigma(K,T)$$

Electronic copy available at: https://ssrn.com/abstract=3307566

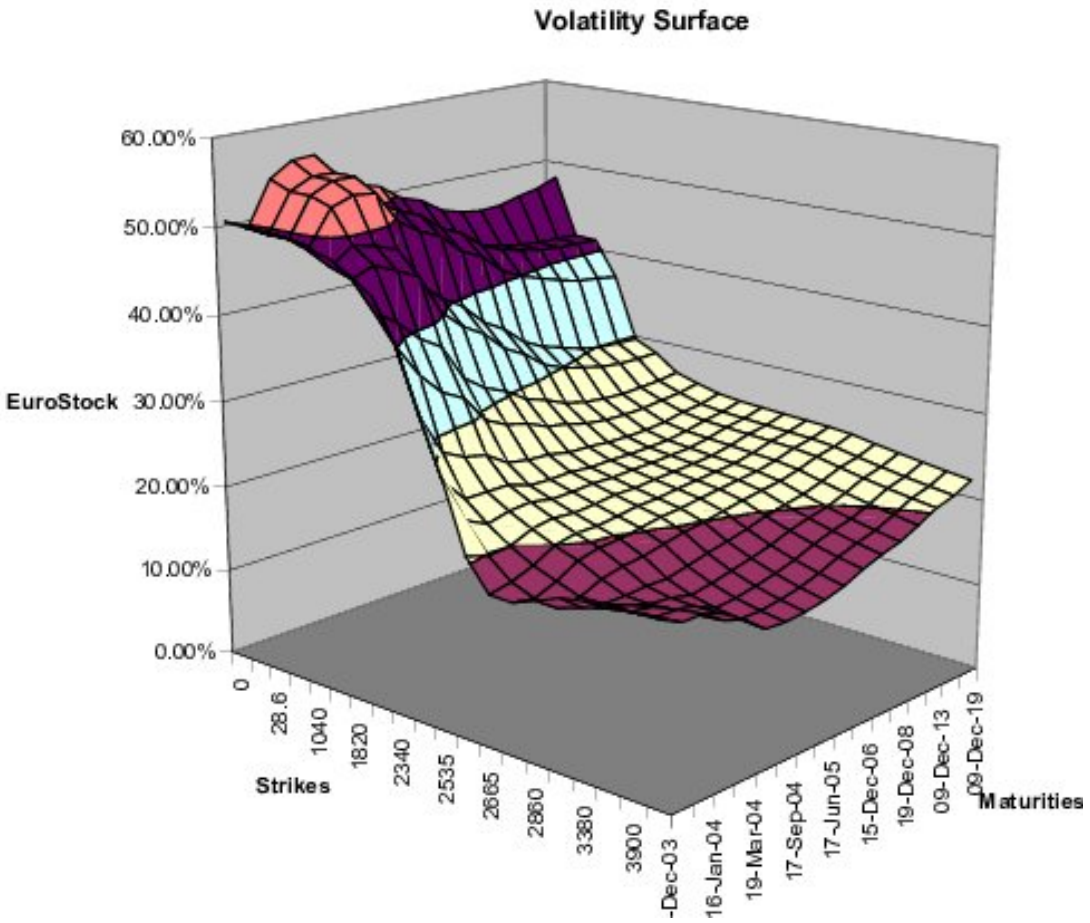


Figure 13.2: EuroStock Volatility Surface on the $8/12/2003$.

as the implied volatility surface. Note, we will some time denote $\Sigma_{BS}(K,T)$ the Black-Scholes implied volatility. A representation of the volatility surface is given in Figure (13.2).
When visualising the IVS, it makes more sense to consider the two-dimensional map $(\eta,T) \to \Sigma(\eta,T)$ [1] where η is the forward moneyness (corresponding to the strike $K = \eta F(t,T)$). We can also use the forward log-moneyness $\overline{\eta}$ such that the two-dimensional map becomes $(\overline{\eta},T) \to \Sigma(\overline{\eta},T)$ [2]. Further, we let the total variance be given by

$$\omega(\eta,T) = \nu^2(\eta,T) = \Sigma^2(\eta,T)(T-t)$$

and let the implied total variance satisfies

$$C(t,S_t,K,T) = C_{TV}(t,S_t,\eta F(t,T),T;\omega(\eta,T)) \tag{13.1.9}$$

where the two-dimensional map $(\eta,T) \to \omega(\eta,T)$ is the total variance surface. Note, $\sqrt{\omega(\overline{\eta},T)}$ corresponds to the time-scaled implied volatility in forward log-moneyness form.

[1] Here $\Sigma(\eta,T)$ must be understood as $\Theta(\eta,T) \to \Sigma(F(t,T)\eta,T)$
[2] Here $\Sigma(\overline{\eta},T)$ must be understood as $\Theta(\overline{\eta},T) \to \Sigma(F(t,T)e^{\overline{\eta}},T)$

Electronic copy available at: https://ssrn.com/abstract=3307566

13.1.1.4 The no-arbitrage conditions

The shape of the implied volatility surface is constrained by absence of arbitrage (AOA) which are as follow:

- we must have $\Sigma(K,T) \geqslant 0$ for all strikes K and maturity T.
- at any given maturity T, the skew can not be too steep (to avoid butterfly arbitrage).
- the term structure of implied volatility can not be too inverted (to avoid calendar spread arbitrage).

Further, understanding the behaviour of the implied volatility (IV) surface for far expiries and far strikes is fundamental for extrapolation problems. Thus, various necessary conditions for an implied volatility surface to be properly defined have been proposed (see Durrleman [2003], Lee [2005]). A summary can be found in textbook by Fengler [2005]. Later, Roper [2010] established necessary and sufficient conditions for an IVS to be free from static arbitrage. These conditions were expressed in terms of implied volatility providing a complete characterisation of an IVS free from static arbitrage (see Rogers et al. [2010]).

Definition 13.1.4 *Arbitrage-Free Implied Volatility Surface*
Assuming deterministic interest and repo rates, an implied volatility surface is free from static arbitrage if and only if the following conditions are satisfied

(A1) (Monotonicity of total variance)
$\forall x$ *and* $T > 0$, $\partial_T \omega(x,T) \geqslant 0$

(A2) (Convexity in K)
$\forall T > 0$, $\partial_{KK} C(K,T) \geqslant 0$

(A3) (Large-Moneyness Behaviour)
$\lim_{\overline{\eta} \to \infty} d_1(\overline{\eta}, \omega(\overline{\eta},t)) = -\infty$ *and* $\lim_{\overline{\eta} \to \infty} d_2(\overline{\eta}, \omega(\overline{\eta})) = -\infty$

(A4) (Small-Moneyness Behaviour)
$\lim_{\overline{\eta} \to -\infty} d_2(\overline{\eta}, \omega(\overline{\eta},t)) = \infty$ *and* $\lim_{\overline{\eta} \to -\infty} d_1(\overline{\eta}, \omega(\overline{\eta})) = \infty$

13.1.1.5 Characterising the smile dynamics

13.1.1.5.1 The regimes of volatility We present some recently proposed deterministic implied volatility models, defined by assuming that either the per-delta or the per-strike implied volatility surface has a deterministic evolution. We then give some rules of thumb used by practitioners to compute their plain-vanilla prices. To do so, we look at some standard evolutions of the implied volatility surface denoted by $\Sigma(t, S_t; K, T)$ and first described by Derman [1999] and extended by various authors such as Daglish et al. [2006]. It provides an indication of some possible behaviour for the smile that one might expect, even though they are not all **arbitrage-free**.

1. sticky delta : the strike of the option is rescaled according to how the current spot evolved with respect to the spot at inception
   $\Sigma(t, S_t; K, T) = \Sigma_0^{obs}(K\frac{S_0}{S_t}, T-t)$
2. absolute floating : the future implied volatility is obtained from the original smile surface by simply reducing time to maturity and linearly offsetting the strike by how much the spot has moved
   $\Sigma(t, S_t; K, T) = \Sigma_0^{obs}(K + (S_0 - S_t), T-t)$
3. absolute sticky (sticky strike) : any dependence on the current spot level or calendar time is ignored
   $\Sigma(t, S_t, K, T) = \Sigma_0^{obs}(K, T-t)$

Electronic copy available at: https://ssrn.com/abstract=3307566

where Σ^{obs} is the current smile. These regimes of volatility were slightly extended by Derman [2008], who for simplicity of exposition, considered the linear smile (only true when the strike is near the money)

$$\Sigma(t_0, S_0; K, T) = a - b(K - S_0) \ , b > 0$$

where a can be seen as the ATM-volatility, and studied its variation when the underying moves away from S_0.

1. sticky moneyness : The dynamics of option prices derive entirely from the moneyness

$$\Sigma(t, S_t; K, T) = a - b(\frac{K}{S_t} - 1)S_0 \ , b > 0$$

   It shifts the skew as the stock price moves by adjusting for moneyness, assuming that the market mean reverts to the value a independently from market level. When the skew is nearly linear we get $\Sigma \approx \Sigma(S_t - K)$, so that the implied volatility rises when S rises. In the BS-delta (see Equation (13.1.4)), the dependence on K and S occurs only through the moneyness η, such that for ATM option we get $|\Delta| = \frac{1}{2}$. However, the delta depends on the scaled moneyness $\frac{1}{\sqrt{\omega(t)}} \log \eta$, which should be the fixed point. In presence of negative skew, the IV increases with increasing spot, leading to a greater delta than BS-delta. Stochastic volatility models and jump-diffusion models have this property (assuming the other variables do not change).

2. sticky implied tree : In the implied tree, local volatility increases as the underlying decreases. Assuming that the IV is approximately the linear average of local volatilities between the initial spot price and the strike price, we can extract the future local and implied volatilities from current implied volatilities. We get the linear approximation to the skew

$$\Sigma(t, S_t; K, T) = a - b(K + S_t - 2S_0) \ , b > 0$$

   so that the implied volatilities decrease as K or S increases, leading to a smaller delta than BS-delta. Hence, when linearly approximating the local volatility we get $\Sigma \approx f(K + S_t)$ and volatilities are inversly correlated with the underlying spot price. The sticky implied tree regime corresponds to a fear of higher market volatility in the case of a fall of the spot price. Local volatility models have this property.

However, these simplified regimes of volatility have a couple of major drawbacks. First of all, they are not consistent with market observable data, and secondly the way they are used to price forward call options is not derived from the theory of No Arbitrage.

13.1.1.5.2 Space homogeneity versus space inhomogeneity Even though the dynamics of implied volatility surface (IVS) are neither stationary nor Markovian, we saw in Section (13.1.1.5.1) that they could be classified in terms of regimes which might change over time. These regimes explain the evolution of the vanilla smiles in terms of the strike price and the underlying spot price. At the two extremes, the smile can either be expressed in terms of moneyness, or in terms of absolute value for the strike and spot prices. The former is generated by space homogeneous models, like the Heston model, and follow the sticky delta rule. The latter is generated by space inhomogeneous models, such as certain types of local volatility models, and follow the sticky implied tree rule. In between, practitioners developed a range of universal models, where the diffusion term is a mix of stochastic volatility and local volatility (see Lipton [2002], Blacher [2001]). The sticky delta rule leads to a delta greater than the BS-delta, meaning that the dynamics of the IV follow that of the underlying asset. In the sticky implied tree rule the delta is smaller than the BS-delta, meaning that the dynamics of the IVS is inversly related to the underlying asset. It is interesting to note that some authors justified their models by the type of dynamics they produce. For Blacher [2001], homogenous smile do not predict a systematic move of the relative smile when spot changes. Observing the equity index, the smile does not move coincidentally with the underlying asset, so that to modify the dynamics of the smile he had to re-introduce inhomogeneity in the stochastic model. On the contrary, for Hagan et al. [2002] local volatility models (LVMs) predict smile dynamics opposite of observed market behaviour. LVMs predict that the smile shifts to higer prices when the underlying asset decreases, and that the smile shifts to lower price when spot price increases. Observing interest rate

Electronic copy available at: https://ssrn.com/abstract=3307566

smiles, where asset prices and market smiles move in the same direction, they devised the SABR model, which is space inomogeneous for $\beta \neq 1$ and allows to capture the correct dynamics of the smile (in the rates market). Thus, we see that the specific model chosen characterise a particular type of dynamics for the IVS, which in turn impose a particular delta hedging ratio.

13.1.2 The calibration problem

13.1.2.1 The general idea

13.1.2.1.1 Description In order to fit the volatility surface practitioners use a range of parametric models, such as jump-diffusion models. For every parametric model that one can define, we need to estimate the model parameters from the implied volatility surface. It leads to an ill-posed inverse problem. It is ill-posed because the inversion is not stable and amplifies market data errors in the solution. To be more precise, letting x be the vector of model parameters and y be the vector of market prices, we want to solve

$$Tx = y$$

where $T : X \to Y$ is a (non-linear) operator between reflexive Banach spaces X, Y, with inverse T^{-1} which is not continuous. We further assume that only noisy data y^δ with

$$||y^\delta - y|| \leqslant \delta$$

is available. The operator T^{-1} can be found by solving a measure among a set of measures, such as

$$||Tx - y^\delta||^2$$

The ill-posness of the problem means that a small error on market data y can lead to a big error on the solution (model parameters) x. Regularisation techniques allow one to capture the maximum information on x from the set y in a stable fashion.

13.1.2.1.2 Measures of pricing errors We choose to estimate implicitly the vector Ψ of model parameters by minimising, given a measure, the distance between the liquid market price $P_t(T_i, K_i)$ and the model price $C_t(T_i, K_i)$ for $i \in I$ where I is the total number of market prices considered. Note, the market price for the ith stock S_i is denoted by $P_i(t)$ while the option price for maturity T_i and strike K_i is denoted by $P_t(T_i, K_i)$. We consider some benchmark instruments with payoff $(H_i)_{i\in I}$ with obsereved market prices $(P_i^*)_{i\in I}$ in the range $P_i^* \in [P_i^b, P_i^a]$ representing the bid/ask prices. In the option world, we also need to consider a set of arbitrage free model $\mathcal{Q}$ such that the discounted asset price $(\overline{S}_t)_{t\in[0,T]}$ is a martingale under each $\mathbb{Q} \in \mathcal{Q}$ with respect to its own history $\mathcal{F}_t$ and

$$\forall \mathbb{Q} \in \mathcal{Q}\ ,\ \forall i \in I\ ,\ E^{\mathbb{Q}}[|H_i|] < \infty\ ,\ E^{\mathbb{Q}}[H_i] = P_i^*$$

Since the market price P_i^* is only defined up to the bid-ask spread we get

$$\forall \mathbb{Q} \in \mathcal{Q}\ ,\ \forall i \in I\ ,\ E^{\mathbb{Q}}[|H_i|] < \infty\ ,\ E^{\mathbb{Q}}[H_i] \in [P_i^b, P_i^a]$$

Further, different choices of norms for the vector $(P_i^* - E^{\mathbb{Q}}[H_i])_{i\in I}$ lead to different measures for the calibration error. For example, we have

$$\begin{aligned}
||P^* - E^{\mathbb{Q}}[H]||_\infty &= \sup_{i\in I} |P_i^* - E^{\mathbb{Q}}[H_i]| \\
||P^* - E^{\mathbb{Q}}[H]||_1 &= \sum_{i\in I} |P_i^* - E^{\mathbb{Q}}[H_i]| \\
||P^* - E^{\mathbb{Q}}[H]||_p &= \Big(\sum_{i\in I} |P_i^* - E^{\mathbb{Q}}[H_i]|^p\Big)^{\frac{1}{p}}
\end{aligned}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

Obviously we need to choose a measure among this set of measures, and traditionally practitioners consider the Price Norm

$$f_1(i) = \left|P_t(T_i, K_i) - C_t(T_i, K_i; \Psi)\right|^2$$

or the Relative Price Norm

$$f_2(i) = \left|\frac{P_t(T_i, K_i) - C_t(T_i, K_i; \Psi)}{P_t(T_i, K_i)}\right|^2$$

13.1.2.1.3 Moments estimation Using market prices, we need to estimate the vector Ψ of model parameters in order to price exotic options. This amount to solving the following inverse problem (see Cont et al. [2002]).

***Problem* 1** *Given prices $C_t(T_i, K_i)$ for $i \in I$, find the vector Ψ of model parameters such that the discounted asset price $\hat{S}_t = e^{-rt}S_t$ is a martingale and the observed option prices are given by their risk-neutral expectations*

$$\forall i \in I\ ,\ C_t(T_i, K_i) = e^{-r(T-t)}E^{\Psi}\left[(S(T_i) - K_i)^+ | S_t = S\right]$$

That is, we need to retrieve the risk-neutral process and not just the conditional densities (European prices) which is equivalent to a moment problem for the process S. If we new option prices for one maturity and all strikes we could compute the risk-neutral distribution of the logarithm price and deduce the model parameters. We would then compute option prices for any other maturity meaning that we can not gain further information from prices of other maturities.

13.1.2.2 The problem

13.1.2.2.1 The choice of a volatility model In complete markets, pricing models such as Dupire's formula (see Dupire [1994]), require the knowledge of European call and put prices for all strikes and all maturities. However, in practice we can only observe a few market prices from standard strikes and maturities. For an index, one can easily obtain more than 100 points (prices or implied volatility) where some of them are listed while the others are produced by brokers. On the other hand, there are only a few points for single stock (from 0 to 30) produced only by brokers. These points are not always convex in their strikes due to transaction costs or non-synchronous trading, and as such do not satisfy the no-arbitrage conditions. Further, they can have wide or narrow spreads depending on the liquidity on the market and the volume traded. Thus, observed prices contain measurements errors so that the observed implied volatility is equal to the true volatility plus an error. As a result, the market is incomplete and there are more than one acceptable price (volatility) surface satisfying the no-arbitrage conditions. It is typically an ill posed problem as there may be either no solution at all or an infinite number of solutions. Hence, multiple risk-neutral distributions can fit the option prices, so that one needs some additional criteria to generate a unique probability distribution function (pdf). To do so, one can either impose a functional form to the probability distribution and estimate its parameters using option data, or, one can choose non-parametric methods obtaining perfect fit to market data (see Jackwerth [1999], Bondarenko [2003] for a review of these methods). However, non-parametric methods are less adapted to the extrapolation problem than the parametric ones, and they tolerate less control over the generated volatility surface. We are therefore going to use a parametric representation of the market call prices in order to smooth the data and get nice probability distribution functions (pdf).

13.1.2.2.2 Accounting for no-arbitrage We saw in Section (13.1.1.4) that Rogers et al. [2010] gave a necessary condition for no-strike arbitrage. These results have provided us with a set of tools and methods to check whether a given volatility parametrisation is free from arbitrage or not. In markets with long maturity products and discrete dividends, it is important for model pricing to obtain a reliable volatility surface satisfying the no-arbitrage constraint not only in space but also in time. One solution is to define a globally arbitrage-free parametric model in time and space.

Electronic copy available at: https://ssrn.com/abstract=3307566

13.1.2.2.3 The standard approach Given market incompleteness, one need to use additional criterion to choose a solution. It means that we need to reformulate the calibration as an approximation problem, for instance minimising the in-sample quadratic pricing error. It corresponds to the empirical risk minimisation (ERM) problem described in Section (8.1).
We choose to estimate the vector Ψ of model parameters implicitly by minimising, given a measure, the distance between the liquid market price $P_t(T_i, K_i)$ and the model price $C_t(T_i, K_i)$ for $i \in I$, where I is the total number of market prices considered. The market generally consider the optimisation problem

$$\begin{aligned} \Psi^* &= \arg\inf_{\Psi} \mathcal{J}(\Psi) \\ \mathcal{J}(\Psi) &= \sum_{i=1}^{n} w_i \left| P_t(T_i, K_i) - C_t(T_i, K_i; \Psi) \right|^2 + \alpha H(\mathbb{Q}, \mathbb{Q}_0) \end{aligned} \tag{13.1.10}$$

where w_i is a weight associated to each market option price $P_t(T_i, K_i)$. The weights are positive and must sum to one. They should reflect our confidence in individual data points which is determined by the liquidity of a given option. The function H is a measure of closeness of the model $\mathbb{Q}$ to a prior $\mathbb{Q}_0$. Many choices are possible for the penalisation function H. To get uniqueness and stability of the solution, the function should be convex with respect to the model parameters. So, the target function is made of two components

1. penalisation function convex in model parameters Ψ
2. quadratic pricing error measuring the precision of calibration

The coefficient α is the regularisation parameter and defines the relative importance of the two terms such that when $\alpha \to 0$ we recover the standard Least Square Error which is no-longer a convex function. If α is large enough, the target function inherits the convexity properties of the penalising function and the problem becomes well-posed. The correct choice of α is important and can not be fixed in advance since its optimal value depends on the data and on the level of error δ one wants to achieve.

13.1.2.2.4 A global optimum In an incomplete market, a deterministic optimisation method will at best locate one of the local minima of the fitting criterion but will not guarantee the global minima and will not acknowledge the multiplicity of solutions of the initial calibration problem. One solution is to solve the optimisation problem by considering a non-linear programming problem under constraints that do not require computing the gradient of the model. To do so, one can use an improved Differential Evolution (DE) algorithm (see Bloch et al. [2011]) handling constraints in a simple and efficient way.

13.1.3 Supervised learning

Clearly, the calibration problem presented in Section (13.1.2) is a function estimation problem, as described in Section (8.1.1), where the function to learn is Tx. As discussed above, one can either impose a functional form to the probability distribution and estimate its parameters using option data, or, one can choose non-parametric methods obtaining perfect fit to market data. Nonetheless, depending on the chosen model, it can be a long and inefficient process due to the illness of the problem.
We saw in Chapter (8) that with sufficient data, we could train a neural network (NN) to learn the best relationship between some inputs and outputs. As a result, we can use that relationship to interpolate / extrapolate the approximated function. Since neural networks can approximate complex nonlinear relations they could be used to infer the market prices of options. Several authors, such as Malliaris et al. [1993], Hutchinson et al. [1994], Anders [1996], Yao et al. [2000b], Bennell et al. [2004], proposed to use neural networks as a nonparametric method for estimating the pricing formula of European options as well as their delta (first derivative with respect to the spot price).
Most of these autors focussed on the multilayer perceptron (MLP) (see Section (4.1.2)) and radial basis function (RBF)

Electronic copy available at: https://ssrn.com/abstract=3307566

(see Section (7.2)). However, with new computational advancements, some authors such as Culkin et al. [2017], Stark [2017] decided to assess more complex deep learning network to learn the BS-formula. In most cases the networks are capable of learning the BS-formula up to high level of accuracy. Further, compared with the BS-formula as benchmark, the networks outperform the latter on real market data.

13.1.3.1 Fitting model prices

Several authors, such as Hutchinson et al. [1994], Culkin et al. [2017], considered solving the calibration problem in Equation (13.1.10) with neural networks. However, rather than directly fitting a network to the market prices $P_{t,i}$ they considered the exercise of learning the prices $C_{t,i}$ generated by a particular model. Thus, given Equation (8.1.3), the modified optimisation problem becomes

$$\begin{aligned} \theta^* &= \arg\min_{\theta} \overline{L}(\theta) \\ \overline{L}(\theta) &= \sum_{i=1}^{n} w_i \left| \widehat{Y}_{i,t} - C_t(T_i, K_i; \Psi) \right|^2 + \alpha H(\theta, \theta_0) \end{aligned}$$

where $\widehat{Y}_{i,t} = h(X_i, \theta)$ is the network with input (or feature) vector X_i and parameter θ, and the labelled output $Y_{i,t} = C_t(T_i, K_i; \Psi)$ is the model price with parameter Ψ. Note, one can select the network of his choice such as a neural network with a given number of hidden layer and a particular activation function. We can choose any loss function $\overline{L}(\cdot)$ described in Section (8.1.2).

***Remark* 13.1.1** *The network is not build from a no-arbitrage argument (see Section (13.1.1.4)).*

In the case of the BS-formula, the input vector X is made of the stock price S, the option strike K, the option maturity T, the interest rate r and the volatility Σ_{BS}. Note, prices can be normalised and expressed with respect to moneyness, in which case one is using the formula in Equation (13.1.6). In that setting, the BS-formula satisfies the no-arbitrage condition.

13.1.3.1.1 The tests Hutchinson et al. [1994] considered radial basis function network, multilayer perceptron (MLP) network and projection pursuit, and generated (with MC) a two-year sample of daily stock and option prices, called the training path. Several paths were generated to obtain a measure of success of the average network pricing formula. The generation of option prices is based on the CBOE rules where the strike prices are multiples of \$5 for stock prices in the range $[25, 200]$. When options expire a new maturity date is introduced, and the two strike prices closest to the current spot price are used. On average there are 81 options per sample path with a total of 6001 data points. It corresponds to six months of futures options on the $S\&P$ 500 (see below). To reduce the number of variables, options are expressed in terms of the moneyness. Two inputs are used: moneyness and time to maturity. The quality of prices and delta-hedging (one day time-step) is assessed and performances measures are the R^2 and the tracking error for the delta hedging performance. An independent testing period of six months is chosen.

Culkin et al. [2017] trained a multilayer feed-forward network to learn the BS-formula. The chosen variables are moneyness, the option maturity, the spot rate, continuous dividend, and the volatility. In order to generate sufficient data for the network to learn the input-output relationship, they defined a range for each of the component of the input vector and computed the model prices by using the BS-formula in Equation (13.1.3). Using a four hidden layer network, they simulated 300000 call option prices and obtained $R^2 = 0.9982$ in the linear regression against the BS-formula in the out of sample.

The same approach can be repeated with different option pricing models such as a stochastic volatility model (see Heston [1993]) or a jump-diffusion model. Since neural networks are function approximator (see Remark (8.1.1))

Electronic copy available at: https://ssrn.com/abstract=3307566

they can learn complex pricing models and be used in a simple gradient descent or a genetic algorithm to estimate the model's parameters Ψ. Rather than using a Monte Carlo engine or a partial differential equation, which is time consuming, we use the network to interpolate the prices.

13.1.3.2 Fitting model prices

As discussed in Section (13.1.2), we want to use a network to solve the calibration problem in Equation (13.1.10) on real market prices. The optimisation problem becomes

$$\begin{aligned}\theta^* &= \arg\min_\theta \overline{L}(\theta) \\ \overline{L}(\theta) &= \sum_{i=1}^n w_i \left|\widehat{Y}_{i,t} - P_t(T_i, K_i)\right|^2 + \alpha H(\theta, \theta_0)\end{aligned}$$

where $\widehat{Y}_{i,t} = h(X_i, \theta)$ is the network with input (or feature) vector X_i and parameter θ, and the labelled output $Y_{i,t} = P_t(T_i, K_i)$ is the market price. In that setting, the input vector X is made of the stock price S, the option strike K, the option maturity T, and the interest rate r.

***Condition* 1** *Given Remark (13.1.1), one must make sure that the market prices are arbitrage-free in time and space.*

13.1.3.2.1 The tests Hutchinson et al. [1994] also tested their approach on the $S\&P$ 500 futures option market and used the BS-formula as a benchmark with a volatility statistically estimated on a sliding window of 60 days. The volatility is estimated as follows:

$$\widehat{\sigma} = \frac{s}{\sqrt{60}} \tag{13.1.11}$$

where s is the standard deviation of the 60 most recent daily returns. The data cover the five year period from Jan87 to Dec91, the options have quarterly expirations with a total of 40 to 50 points on a typical day. The data is divided into 10 nonoverlapping six-month subperiods, which gives an average of 137 call options and a total number of points of 6246 for a subperiod. A separate learning network is trained on each of the first 9 subperiods and tested on the following subperiod. That is, we keep changing model for each subperiod, or equivalently we re-calibrate the model every six month.

Stark [2017] applied the same methodology on options written on the DAX 30 from 2013 to 2017 and obtained consistent results with previous literature. The dataset contains 668 different call options with 25 different strikes, giving a total of 130132 points. The data has been filtered, options with maturity larger than two years were discarded, moneyness is in the range $[0.7, 1.20]$, which resulted in 48486 points remaining. The total number of days is 1231, the training set is 785 days and the testing set is 424 days. The model is an MLP with four hidden layers and softplus activation function. There are two inputs: the moneyness and time to maturity. Rates and volatility are estimated as in Hutchinson et al.

13.1.3.2.2 Results In general the NNs do better for out-of-the money (OTM) because BS-formula overprices these options. While both NNs and BS-formula underprice in-the-money (ITM) options, the former outperform the latter. Note, near-the-money (NTM) options is not as good as the wings for NNs. When partitioning with respect to maturity NNs are superior to the BS-formula for medium and long term maturities, except for short term maturities at all moneyness level. In that case the NNs overprice the short-term NTM and OTM options. The longer the maturity, the better for the NNs.
When it comes to assessing the delta it seems the NNs outperform the BS-formula for OTM options, but the reverse is true in the case of ITM options. This is not surprising given Remark (13.1.1).

Electronic copy available at: https://ssrn.com/abstract=3307566

13.1.3.2.3 Analysis and solutions Some of the reasons explaining these results are:

- The BS-volatility in Equation (13.1.11) is an estimate of the implied ATM-volatility for short maturities. So naturally the NTM options with short maturities are well approximated with the BS-formula. As the time to maturity increases and one consider the wings of the surface this volatility no-longer matches the IVS, explaining the poor results of the BS-formula.
- It has been assumed that the distribution of the returns is independent of the level of the spot price, so that the pricing formula is homogeneous of degree one both in the spot level and strike price. However, as discussed in Section (13.1.1.5), it could be homogeneous, inhomogeneous, or anything in between.
- The pricing function is not smooth near-the-money.
- The approximation of the derivative of the pricing formula require to impose smoothness constraints.
- Even though neural networks are function approximator (see Remark (8.1.1)), it does not guarantee that the derivatives of the function are also learned. Need to impose smoothness constraints on the loss function.

Some solutions to solve these problems are:

- A measure of performance for validating the capacity of a network to reproduce market prices should be to check that the option prices generated in the testing sample are within the market bid-ask.
- One should relax the homogeneous assumption and consider the spot level and strike price as variables.
- To refine the granularity of the variables, especially around the money, where the delta goes from convex to concave shape. Thus, if we want to cover large range of strikes we need to massively increase the size of the data.
- Alternatively, one can keep coarse points but learn the delta of the market prices.

This is what we are going to discuss in the next section.

13.1.3.3 Completing market prices

13.1.3.3.1 The problem Contrary to fitting a theoretical model, at a given time t, we only observe a finite number of market prices (usually very small). Even when considering option prices at different time t, Hutchinson et al. [1994] only gathered around 6000 options on the $S\&P$ 500 over a period of six months. This is more than unsufficient to train a complex multilayer model. Further, chances are that the spot price S does not vary too much over such a period, so that we can only observe prices around the moneyness (see Equation (13.1.5)). However, over a longer period of time the moneyness will vary and possibly jumps (October 1987 crash, subprime crisis in 2008). Acknowledging this problem, Hutchinson et al. [1994] considered changing the learning network every six month. However, in order to get good learning results without changing (or recalibrating) the network, the sample under consideration should span all the domain of definition. That is, it should cover the moneyness in a range of say $[0.1, 2.0]$ for all liquid maturities. Put another way, since we do not know the joint probability distribution $P(X, Y)$, we need to generate a large number data spanning all the domain of definition, that is, European call and put prices for all strikes, all maturities, and all spot level.

13.1.3.3.2 Generating arbitrage-free data Unfortunately there is no exact model we can use to generate sufficient market prices. Nonetheless, to generate a continuum of market prices, we can use a parametric representation of the market call prices that smooth the data and get nice probability distribution functions (pdf). However, we saw in Section (13.1.1) that we needed to introduce necessary conditions to obtain an implied volatility surface (IVS) free from static arbitrage. One solution is to define a globally arbitrage-free parametric model in time and space capable

Electronic copy available at: https://ssrn.com/abstract=3307566

of reproducing a large range of implied volatility surfaces. To do so, Bloch [2010] proposed to impose smoothness and value constraints directly on the market prices and their resulting implied volatility surface. He considered a parametric representation of the market call prices under constraints called MixVol model (see Appendix (13.4)) in order to smooth the data and get nice probability distribution functions (pdf). A differential evolution algorithm (see Bloch et al. [2011]) was used to calibrate the model's parameters to a finite set of option prices. We can therefore fit the existing market prices with this model and use it to interpolate / extrapolate market prices in time and space in an arbitrage free way.
However, if we want to produce hundred of thousands of market prices for all strikes, all maturities, and all spot level we would need a very large history of option prices to calibrate. Alternatively, if we had an intuitive parametric representation of the smile allowing us to manually modify its shape when the underlying price moves, while preserving the no arbitrage in time and space, we could generate sufficient data. This is precisely what we do in the MixVol model when we run the scenarios analysis described in Appendix (13.4.3). In addition, as described in Appendix (13.4.2), we can compute analytically the Greeks of the model. Thus, we can generate:

- European call and put prices for all strikes, all maturities, and all spot level.
- Their associated Greeks

As a result, we are guaranteed to generate sufficient data to properly train a deep network to learn both market prices and their associated Greeks, while satisfying the Condition (1).

13.1.4 Reinforcement learning

13.1.4.1 Introduction

13.1.4.1.1 Problems with continuous models We saw in Section (13.1.1.1) that the BS-formula was derived from the assumption that market returns were i.i.d. random variables and that the option price could be continuously hedged. That is, the continuous-time limit is assumed so that pricing by replication becomes exact in this limit, and all risks are instantaneously eliminated. Taking the limit, the option becomes risk-less, but also redundant. While this limit makes mathematical sense, it is note the case in practice. Nonetheless, this model became the foundation of mathematical finance.
Continuous-time models developed where stochastic calculus is used to model the infinitesimal movements of the underlying asset. Denoting dX a Wiener process, we use Ito's lemma to understand its behaviour in the limit $dt \to 0$. It relates the small change in a function of a random variable to the small change in the random variable itself. A heuristic approach to Ito's lemma is based on the Taylor series expansion. As a short cut one consider the following result: with probability one, $dX^2 \to dt$ as $dt \to 0$. Some textbooks even display multiplication table (see Table (13.1)).

	dt	$dX_{i,t}$	$dX_{j,t}$
dt	0	0	0
$dX_{i,t}$	0	dt	0
$dX_{j,t}$	0	0	dt

Table 13.1: Multiplication table

However, in practice, prices are quoted at discrete-time intervals Δt. As a result, traders use the BS-formula to compute the market prices, but approximate their delta (the Greeks) to make them discrete. For instance, one uses central difference (or other schemes) with a particular space step ΔX. As such it becomes model dependent (see Benhamou [2001]). That is, we need to assume something about the dynamics of the market prices in the discrete-time interval Δt for the space step ΔX. One solution is to consider the regimes of volatility described in Section (13.1.1.5), but

Electronic copy available at: https://ssrn.com/abstract=3307566

the resulting dynamics are not arbitrage-free. Another solution is to complexify the dynamics of the underlying asset such as using stochastic volatility model. However, in a large enough discrete-time interval Δt, the space step ΔX may no-longer be modelled with an Ito's process, which describes infinitesimal movements of the spot prices. This is problematic in the case of market shocks, when the market prices exhibit large movements. One solution is to modify the dynamics of the underlying asset by adding jumps, but the resulting prices are not built from the replicating strategies.

All the problems associated with the use of continuous-time models to price and hedge options led market risk professionals to come up with some methods to understand the dichotomy between market prices and the cost of replication of these options. One of this solution is the profit and loss (PnL) explain which consist in decomposing the price of an option with Taylor expansion series [3], but with arbitrary discrete-time intervals Δt and space steps ΔX. The aim being to approximate the option price with a finite number of terms (Taylor polynomial). This is a heuristic way of measuring the error of discretely hedging an option while pricing it with a continuous model. Obviously, the risk occurs in presence of large movements from the underlying asset. The larger the discrete-time interval Δt, the larger the risk. As such, there is no reason for the PnL explain to be close to zero unless the space steps ΔX are infinitesimal during that period of time. Nonetheless, the experts from market risk do everything in their power to make the PnL zero. That is, while the traders use a continuous model to price and hedge the option, they try to come up with a way of explaining how this option would move in that discrete time-interval given the market space step ΔX (which can be non-negligible).

13.1.4.1.2 Accounting for risk Since intrinsic risk is at the heart of option pricing (see Appendix (13.3.1)), the pricing formula should consider a discrete time setting. However, in such a setting one has to consider all the transition probabilities and their implication to the dynamics of market prices. That is, the Kolmogorov-Compatibility condition (see Appendix (13.3.4)) states that when the number of fixing dates in a model is finite, there is an infinity of conditional densities. In continuous time models with volatility a deterministic function of time and stock price, such as the Dupire model, the volatility surface is assumed Markovian and stationary, giving a unique solution to the conditional densities and hence to the forward volatility. However, the implied volatility surface is neither stationary nor Markovian, but stochastic.
One solution is to consider complex continuous models such as jump-diffusion models or to relax the continuity assumption and deal with the conditional densities. Some authors such as Follmer et al. [1989] and Schweizer [1995] proposed some incomplete market models, where hedging is presented as a quadratic risk minimisation problem.
In the case of interest to us we will consider a sequential risk minimisation in a Markov decision process (MDP) problem. In general, we assume the transition matrix and the expected next step reward to be known and use dynamic programming to solve the problem. However, we can relax this assumption and use reinforcement learning (RL) (see Chapter (9)) to solve this problem. In that case the agent learns from data generated from an option trading strategy. Gosavi [2010] proposed the construction of a risk-sensitive MDP by adding a one-step variance penalties to a finite-horizon risk-neutral MDP problem (see Definition (9.3.2). He used a CVAR-adjusted price as a compensation for the risk in the option. In the same spirit, assuming normal returns, Halperin [2017] proposed to follow a multi-period version of the mean-variance approach (see Appendix (13.3.3)) to derive a discrete BS-formula in incomplete market. He incorporated a quadratic reward into a standard MDP formulation, obtaining the QLBS model. Halperin [2018] provided a numerical Q-learning analysis of his model.
Interestingly, the continuous-time BS-model is shown to be the continuous-time limit of a model-based RL (in the case of normal returns), where all the data needed are: the current stock price and the volatility. In the discrete case, the pricing formula requires a large quantity of data. In both cases, we need a lot of data to either learn in a supervised way the relation between the stock-volatility and the market prices, or the optimal hedging strategy.
We extend this approach by considering non-normal market returns and show that the reward depends on the variance

[3] A Taylor series is a representation of a function as an infinite sum of terms that are calculated from the values of the function's derivatives at a single point. The Taylor series requires the knowledge of the function on an arbitrary small neighbourhood of a point. That is, the error is very small in a neighbourhood of the point where it is computed, while it may be very large at a distant point.

Electronic copy available at: https://ssrn.com/abstract=3307566

of the portfolio as well as its skewness and kurtosis. We are going to describe these approaches in light of the results on RL (see Chapter (9)) as well as the CAPM (see Appendix (13.3.3)).

13.1.4.2 Deterministic BS-model

13.1.4.2.1 Pricing in discrete time We follow the notation on Appendix (13.3.4) and consider the seller of a European option (put option) with maturity T and payoff $H = h(X(T))$. The replicating portfolio at time t is $V(t)$, the admissible trading strategy is (α, β) and $P(t,T) = e^{-r(t-t)}$ is the zero-coupon bond price. We want to devise a pricing formula that account for intrinsic risk. We choose to minimise the risk of the portfolio by hedging in a sequential decision process. We follow the local risk minimisation approach (see Follmer et al. [1989]). In the mean-variance approach introduced by Markowitz in the portfolio allocation problem (see Appendix (13.3.3)), the risk at time t is given by

$$R_t = \lambda Var_t(V(t))$$

where the parameter λ is a scalar. In that setting, the risk of the portfolio is expressed in terms of its variance. Following the method presented in Appendix (13.3.2), the risk-averse price for the seller of the option at time $t = 0$ is given by

$$C^{ask}_{t=0}(X, \alpha) = E_0[V_0 + \lambda \sum_{t=0}^{T} P(0,t) Var_t(V(t)) | X_0 = X, \alpha_0 = \alpha] \tag{13.1.12}$$

where λ is the risk tolerance (or risk premium) and the second term is the discrete sum of the discounted risk (variance) of a hedge portfolio from all future hedge rebalance periods. This choice of risk premium for option pricing is not unique. We can reformulate this problem as the maximisation of the objective function

$$V(X, \alpha) \equiv -C^{ask}_{t=0}(X, \alpha)$$

This is a stochastic control problem where the quantity of the asset held α is the control, and the asset X is the state of the system. Assuming normal returns, Halperin [2017] proposed a negative quadratic reward to rewrite the pricing formula in terms of the value function of a finite-horizon MDP problem.

13.1.4.2.2 The value function We need to express the pricing formula in Equation (13.1.12) in terms of the value function to use the techniques developed in reinforcement learning. That is, we need to identify the reward recovering the option payoff at maturity.
A time-homogeneous process X_t is defined with dynamics

$$dX_t = \sigma dW_t$$

where σ is a constant volatility, W_t is a Brownian motion. Integrating in the range $[t, t+1]$, we get

$$\Delta X_t = X_{t+1} - X_t = \sigma \Delta W_t$$

The value of the time-dependent stock price is given by

$$S_t = e^{X_t + \left(\mu - \frac{\sigma^2}{2}\right)t}$$

where μ is a drift. This is the BS-model with constant volatility and normally distributed returns. The variation in the stock price is given by

$$\Delta S_t = S_{t+1} - S_t = S_t\left(e^{\left(\mu - \frac{\sigma^2}{2}\right) + \sigma \Delta W_t} - 1\right)$$

We denote the action $a_t = a_t(X_t)$ in terms of X and $u_t = u_t(S_t)$ in terms of S. They are related as follows:

Electronic copy available at: https://ssrn.com/abstract=3307566

$$u_t = a_t\big(\log S_t - (\mu - \frac{\sigma^2}{2})t\big)$$

A time dependent policy $\pi(t, X_t)$ is introduced, where $\pi : \{0, ..., T-1\} \times \mathcal{X} \to \mathcal{A}$ is a deterministic policy mapping at time t the state $X_t = x_t$ into the action $a_t \in \mathcal{A}$ given by $a_t = \pi(t, x_t)$. The pricing formula in Equation (13.1.12) can be rewritten in terms of the value function as follows:

$$V_{\pi,t}(X_t) = E_t[-V(t) - \lambda Var(V(t)) - \lambda \sum_{t'=t+1}^{T} P(t, t^{'})Var_{t'}(V(t^{'}))|\mathcal{F}_t]$$

Given the value function with a shifted time argument:

$$V_{\pi,t+1}(X_{t+1}) = E_{t+1}[-V(t+1) - \lambda \sum_{t'=t+1}^{T} P(t+1, t^{'})Var_{t'}(V(t^{'}))|\mathcal{F}_{t+1}]$$

we get

$$-\lambda E_{t+1}[\sum_{t'=t+1}^{T} P(t+1, t^{'})Var_{t'}(V(t^{'}))|\mathcal{F}_{t+1}] = V_{\pi,t+1}(X_{t+1}) + E_{t+1}[V(t+1)|\mathcal{F}_{t+1}]$$

We can therefore write

$$\begin{aligned} -\lambda E_{t+1}[\sum_{t'=t+1}^{T} P(t, t^{'})Var_{t'}(V(t^{'}))] &= -\lambda E_{t+1}[\sum_{t'=t+1}^{T} P(t, t+1)P(t+1, t^{'})Var_{t'}(V(t^{'}))] \\ &= \gamma\Big(V_{\pi,t+1} + E_{t+1}[V(t+1)]\Big), \gamma \equiv P(t, t+1) \end{aligned}$$

Plugging it back in the above equation, using the recursive Equation (13.3.16) (and changing notation) and the conditional expectation (see Theorem (14.6.5)), we get the value function

$$\begin{aligned} V_{\pi,t}(X_t) &= E_t[-V(t) - \lambda Var(V(t)) + \gamma\Big(V_{\pi,t+1} + E_{t+1}[V(t+1)]\Big)|\mathcal{F}_t] \\ &= E_t[-\gamma\Big(V(t+1) - a_t\Delta S(t)\Big) - \lambda Var(V(t)) + \gamma\Big(V_{\pi,t+1} + E_{t+1}[V(t+1)]\Big)|\mathcal{F}_t] \\ &= E_{\pi,t}[r(X_t, a_t, X_{t+1}) + \gamma V_{\pi,t+1}(X_{t+1})] \end{aligned}$$

where the one-step time-dependent reward is

$$\begin{aligned} r_t = r(X_t, a_t, X_{t+1}) &= \gamma a_t \Delta S_t(X_t, X_{t+1}) - \lambda Var_t(V(t)), t = 0, ..., T-1 \\ &= \gamma a_t \Delta S_t(X_t, X_{t+1}) - \lambda\gamma^2 E_t[\big(\widehat{V}(t+1)\big)^2 - 2a_t\Delta\widehat{S}_t\widehat{V}(t+1) + a_t^2\big(\Delta\widehat{S}_t\big)^2] \end{aligned}$$

where $\Delta S_t(X_t, X_{t+1})$ emphasis the fact that it is a function of X_t and X_{t+1}, $\widehat{V}(t+1) \equiv V(t+1) - \overline{V}(t+1)$ and $\overline{V}(t+1)$ is the sample mean of all values of $V(t+1)$, and similarly for $\Delta\widehat{S}_t$. At time $t = T$ we get $r_T = -\lambda Var(V(T))$. So the expected reward at time t is quadratic in the action a_t, that is,

$$E_t[r(X_t, a_t, X_{t+1})] = \gamma a_t E_t[\Delta S_t] - \lambda\gamma^2 E_t[\big(\widehat{V}(t+1)\big)^2 - 2a_t\Delta\widehat{S}_t\widehat{V}(t+1) + a_t^2\big(\Delta\widehat{S}_t\big)^2]$$

The optimal policy maximises the value function

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\pi_t^*(X_t) = \arg\max_{\pi} V_{\pi,t}(X_t)$$

so that the optimal value function satisfies the BOE.
Note, the return r_t corresponds to the expected growth rate for the period between time t and $t+1$ (see Appendix (13.3.3)). That is, it corresponds to the mean-variance criterion for normal returns, where the option is replicated with a quantity of the spot price and some cash. Since, this selection is repeated at each time step, the MDP problem is a multi-period version of the portfolio selection problem.

13.1.4.2.3 The Q-value function We can also consider the Q-value function with the initial state-action pair (s,a) (see Definition (9.3.4)). Its value at time t is given by

$$Q_{\pi,t}(s,a) = E_t[-V(t,s_t)|s_t = s, a_t = a] - \lambda E_{\pi,t}[\sum_{t'=t}^{T} P(t,t^{'})Var_{t'}(V(t^{'},s_{t'}))|s_t = s, a_t = a]$$

The first expectation only involves averaging over the next time step and as such does not depend on the policy π. Since we can write the value function $V(s)$ in terms of the Q-value function (see Equation (9.3.12)), the Bellman equation for the Q-value function (9.3.13) is

$$Q_{\pi,t}(s,a) = E_t[r_t|(s,a)] + \gamma E_{\pi,t}[V_{\pi,t+1}(s_{t+1})|s]$$

An optimal Q-value function $Q_{\pi,t}^*(s,a)$ is obtained when the Q-value function is evaluated with an optimal policy π_t^* (see Definition (9.3.5)). That is,

$$\pi_t^* = \arg\max_{\pi} Q_{\pi,t}(s,a)$$

The optimal value function and Q-value function are:

$$\begin{aligned} V_t^*(s) &= \max_a Q_t^*(s,a) \\ Q_t^*(s,a) &= E_t[r_t|(s,a)] + \gamma E_t[V_{t+1}^*(s_{t+1})|s] \end{aligned}$$

Plugging the value function in the Q-value function, we get the BOE (see Equation (9.3.15))

$$Q_t^*(s,a) = E_t[r_t + \gamma \max_{a_{t+1}\in\mathcal{A}} Q_{t+1}^*(s_{t+1},a_{t+1})|s_t = s, a_t = a], t = 0,...,T-1$$

with terminal condition

$$Q_t^*(s_T, a_T = 0) = -V(T,s_T) - \lambda Var(V(T,s_T))$$

We want a greedy policy π^* that always seeks an action maximising the Q-value function in the current state:

$$\pi_t^*(s_t) = \arg\max_{a_t\in\mathcal{A}} Q_t^*(s_t,a_t)$$

13.1.4.3 Learning the optimal policy

We can then use some of the methods described in Section (9.4) to solve this optimisation problem. It can be solved with dynamic programming (DP), or, we can also use reinforcement learning (RL). As discussed in Section (9.4), dynamic programming assumes that we know the transition matrix of the environment $P(s^{'}|s,a)$ as well as the expected next step reward. We can then use methods such as Value Iteration and Temporal Difference (TD) learning, among others. In the case where the transition probabilities and the reward function are not known we can directly rely on the

Electronic copy available at: https://ssrn.com/abstract=3307566

data to find an optimal policy and use methods such as Fitted Value Iteration, the Q-learning, or the policy gradients, among others. However, in that case, the pricing formula is not built from the no-arbitrage theory and may not satisfies the Kolmogorov-Compatibility condition (see Appendix (13.3.4)).

13.1.4.3.1 Known model Substituting the expected reward (quadratic form) into the BOE, the optimal value $Q_t^*(s_t, a_t)$ is quadratic in the action variable a_t. In that setting, the optimal action is computed analytically

$$a_t^*(s_t) = \frac{E_t[\Delta\widehat{s}_t\widehat{V}(t+1) + \frac{1}{2\gamma\lambda}\Delta s_t]}{E_t[\left(\Delta\widehat{s}_t\right)^2]}$$

As a result, using this analytical action, the policy optimisation in the Value Iteration method is performed analytically:

$$Q_t^*(s_t, a_t) = \gamma E_t[Q_{t+1}^*(s_{t+1}, a_{t+1}^*) - \lambda\gamma\left(\widehat{V}(t+1)\right)^2 + \lambda\gamma\left(a_t^*(s_t)\right)^2\left(\Delta\widehat{s}_t\right)^2], t = 0, ..., T-1$$

Thus, in this particular case, no discretisation of the action space $\mathcal{A}$ is required due the quadratic dependence of the Q-value function Q_t on the action a_t at time t. The action in this equation is always just one optimal action. Thus, the fair ask option price is $C_t^{QLBS} = -Q_t(s_t, a_t^*)$. The whole calculation is semi-analytic.
In the case of continuous-space scenarios with MC simulations, the optimal hedge and Q-function are expanded over some set of basis functions and the results are expressed in terms of the expansion coefficients. Rather than using a lattice, which applies to a finite-state version of the model, the MC is more general as it average over all scenarios from time $[t, t+1]$ simultaneously. Therefore, there is no need in explicit conditioning on s_t at time t. Learning optimal actions for all states simultaneously means learning a policy (see Remark (9.3.5)), and as such it applies to RL for unknown model.

13.1.4.3.2 Unknown model In the cases where transition probabilities and rewards are not known, the Bellman optimality equation (BOE) is solved recursively by using samples. Since the reward function is quadratic, the author obtained semi-analytical solutions by using the Q-learning method. However, to guarantee convergence to the optimal hedge we need enough training data.

Given some historical data and no access to a real-time environment, the author considered a batch-mode learning with a set of N trajectories and information set $\mathcal{F}_t = \{\mathcal{F}_t^{(n)}; n = 1, ..., N\}$, which contains the asset (or state) s_t, the hedge position a_t, the reward r_t and the next-time state s_{t+1}. That is,

$$\mathcal{F}_t^{(n)} = \{\left(s_t^{(n)}, a_t^{(n)}, r_t^{(n)}, s_{t+1}^{(n)}\right); t = 0, .., T-1\}$$

The dynamics are assumed to be Markov so that we have $N \times T$ single-step transitions $P(s^{'}|s, a)$ given. Note, some data is artificial to track the performance of a hypothetical stock and cash replicating portfolio for a given option. The value function can be decomposed into immediate reward plus discounted value of successor states (see Equation (9.3.8)). The Bellman equation at time t is

$$V_{\pi,t}(s_t) = E_{\pi,t}[r_t + \gamma V_{\pi,t+1}(s_{t+1})]$$

where $r_t = r(s_t, a_t, s_{t+1})$ is a one-step time-dependent random reward. In that setting, the risk-adjusted portfolio return, or, the expected growth rate, is given by

$$r_t = \gamma a_t \Delta s_t - \lambda Var_t(V(t))$$

where $\Delta s_t = s_{t+1} - s_t$.
The author used the Fitted Q-Iteration method to solve this problem, which combines the Q-learning method with function approximations to deal with large state spaces (see Section (9.4.2)). Convergence of the Q-learning method (see Theorem (9.4.1)) requires an exhaustive sampling of the state-action pairs (s, a). Even if the training data (in the

Electronic copy available at: https://ssrn.com/abstract=3307566

form of tuples) comes from an option trading desk it might not be sufficient to obtain convergence (asymptotically large data set) and one might have to artificially simulate some data. Note, the actions can be completely random and the Q-learner will still learn the correct price and hedge provided that there is sufficient data. Further, since optimal hedges are obtained using cross-sectional information across all MC paths, such information could be masked in the updating process. In the case of interest to us, the update of the Q-values and actions a_t^* at all points on the grid is done simultaneously by looking at each time t at a snapshot of all MC paths. This is because in the batch-mode learning all the data is already there.
More sophisticated learning algorithms could be used such as deep RL, policy gradients methods, and actor-critic algorithms (see Section (9.4)).

13.1.4.4 Non-normal rates model

13.1.4.4.1 The assumption of normal returns There are two parts to the pricing and hedging of the deterministic BS-model presented in Section (13.1.4.2):

1. The hedging: minimising the portfolio variance across all simulated MC paths.
2. The pricing: adding the cumulative expected discounted variance of the hedge portfolio along all time steps with a risk-aversion parameter λ. Then minimise this option price.

In both cases market risk is expressed in terms of portfolio variance, and in both cases we want to minimise that risk. Hence, hedging and pricing is consistent, so that we get a single computable formula for pricing and hedging options. However, the variance is only a measure of risk in the special case of Gaussian processes. Higher moments need to be taken into consideration in the case of non-normal returns. In some models the variance is not even defined.
The MDP problem is clearly model dependent because the pricing and hedging formula have been derived for normal market rates. As such the computation of the reward in the MDP problem is quadratic in the action, leading to analytical solution to the optimal policy.

13.1.4.4.2 Beyond normal returns Note, this method of pricing and hedging options corresponds to the utility-based approach, where the quadratic utility function has been chosen. It is generally used for its excellent tractability properties, but a risk averse investor with a quadratic utility will increase the percentage of his wealth invested in risky assets as his wealth increases.
The introduction of portfolio variance as a measure of risk in both the pricing and hedging formulae is consistent with the mean-variance criterion in the CMAP approach of portfolio selection (see Appendix (13.3.3.1)) when returns are normally distributed.
It comes from the fact that an exponential utility $U(x) = -e^{-\gamma x}$ can be approximated to give a quadratic certain equivalent (CE) of the portfolio. Taking the limit $\gamma \to 0$ and making a correction via an expansion in powers of γ, an approximate relation between the parameter λ of the quadratic minimisation and the parameter γ of the exponential utility is given by $\lambda \approx \frac{1}{2}\gamma$ (see Appendix (13.3.3.1)).

The optimal hedge α^* corresponds to solving the optimisation problem in Equation (13.3.17), that is, minimising the portfolio variance when the drift μ is a constant. However, we are implicitly assuming that returns are normally distributed. When returns are non-normal, the certain equivalent (CE) of the portfolio also depends on skewness and kurtosis.

13.1.4.4.3 The modified risk-averse price In the case of non-normal returns, a risk averse investor having an exponential utility has aversion to risk associated with increasing variance, negative skewness, and increasing kurtosis. Accounting for these risk, the optimal hedge at time t should be

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned}\alpha_t^* &= \arg\min_{\alpha}\big(Var_t - S_t + K_t\big) \\ &= \arg\min_{\alpha}\Big[Var_t\big(V(t+1)-\alpha(t)\Delta X(t)\big) - S_t\big(V(t+1)-\alpha(t)\Delta X(t)\big) + K_t\big(V(t+1)-\alpha(t)\Delta X(t)\big)\Big] \\ &\quad , t = T-1, ..., 0\end{aligned}$$

where $Var_t = Var_t(V(t))$ is the variance of the portfolio, $S_t = S_t(V(t))$ is the skewness, and $K_t = K_t(V(t))$ is the kurtosis. Obviously, this is no-longer a quadratic minimisation problem.
Similarly, we can build a modified risk-averse price by adding the cumulative expected discounted variance of the hedge portfolio, subtracting the cumulative expected discounted skewness of the hedge portfolio, and adding the cumulative expected discounted kurtosis of the hedge portfolio. In that case, the modified pricing formula becomes

$$\widetilde{C}_0^{ask}(X,\alpha) = E_0[V(0)+\lambda_V\sum_{t=0}^{T} e^{-rt}Var_t(V(t))-\lambda_S\sum_{t=0}^{T} e^{-rt}S_t(V(t))+\lambda_K\sum_{t=0}^{T} e^{-rt}K_t(V(t))|X(0)=X,\alpha_0=\alpha]$$

where λ_V is the risk aversion parameter for the variance, λ_S is the risk aversion parameter for the skewness, and λ_K is the risk aversion parameter for the kurtosis.

13.1.4.4.4 The modified value function Changing notation, we can reformulate this problem as the maximisation of the objective function

$$V(S,a) \equiv -\widetilde{C}_{t=0}^{ask}(S,a)$$

We need to define a dynamics of the process X as well as a transformation function to express the stock price S in terms of the process X. We will not describe this now.
The modified pricing formula can be rewritten in terms of the value function as follows:

$$V_{\pi,t}(X_t) = E_t[-V(t)-\lambda_V Var_t+\lambda_S S_t-\lambda_K K_t-\lambda_V\sum_{t'=t+1}^{T} P(t,t')Var_{t'}+\lambda_S\sum_{t'=t+1}^{T} P(t,t')S_{t'}-\lambda_K\sum_{t'=t+1}^{T} P(t,t')K_{t'}|\mathcal{F}_t]$$

Given the value function with a shifted time argument:

$$V_{\pi,t+1}(X_{t+1}) = E_{t+1}[-V(t+1)-\lambda_V\sum_{t'=t+1}^{T} P(t+1,t')Var_{t'}+\lambda_S\sum_{t'=t+1}^{T} P(t+1,t')S_{t'}-\lambda_K\sum_{t'=t+1}^{T} P(t+1,t')K_{t'}|\mathcal{F}_{t+1}]$$

we get

$$\begin{aligned}&-\lambda_V E_{t+1}[\sum_{t'=t+1}^{T} P(t+1,t')Var_{t'}|\mathcal{F}_{t+1}] + \lambda_S E_{t+1}[\sum_{t'=t+1}^{T} P(t+1,t')S_{t'}|\mathcal{F}_{t+1}] \\ &-\lambda_K E_{t+1}[\sum_{t'=t+1}^{T} P(t+1,t')K_{t'}|\mathcal{F}_{t+1}] = V_{\pi,t+1}(X_{t+1}) + E_{t+1}[V(t+1)|\mathcal{F}_{t+1}]\end{aligned}$$

We can therefore write

Electronic copy available at: https://ssrn.com/abstract=3307566

$$
\begin{aligned}
& -\lambda_V E_{t+1}[\sum_{t'=t+1}^{T} P(t,t')Var_{t'}] + \lambda_S E_{t+1}[\sum_{t'=t+1}^{T} P(t,t')S_{t'}] - \lambda_K E_{t+1}[\sum_{t'=t+1}^{T} P(t,t')K_{t'}] \\
= & -\lambda_V E_{t+1}[\sum_{t'=t+1}^{T} P(t,t+1)P(t+1,t')Var_{t'}] + \lambda_S E_{t+1}[\sum_{t'=t+1}^{T} P(t,t+1)P(t+1,t')S_{t'}] \\
& -\lambda_K E_{t+1}[\sum_{t'=t+1}^{T} P(t,t+1)P(t+1,t')K_{t'}] \\
= & \gamma\Big(V_{\pi,t+1} + E_{t+1}[V(t+1)]\Big), \gamma \equiv P(t,t+1)
\end{aligned}
$$

Plugging it back in the above equation, and using the recursive Equation (13.3.16) (and changing notation), we get the value function

$$
\begin{aligned}
V_{\pi,t}(X_t) &= E_t[-V(t) - \lambda_V Var_t + \lambda_S S_t - \lambda_K K_t + \gamma\Big(V_{\pi,t+1} + E_{t+1}[V(t+1)]\Big)|\mathcal{F}_t] \\
&= E_t[-\gamma\Big(V(t+1) - a_t \Delta S(t)\Big) - \lambda_V Var_t + \lambda_S S_t - \lambda_K K_t + \gamma\Big(V_{\pi,t+1} + E_{t+1}[V(t+1)]\Big)|\mathcal{F}_t] \\
&= E_{\pi,t}[r(X_t, a_t, X_{t+1}) + \gamma V_{\pi,t+1}(X_{t+1})]
\end{aligned}
$$

where the one-step time-dependent reward is

$$
r_t = r(X_t, a_t, X_{t+1}) = \gamma a_t \Delta S_t(X_t, X_{t+1}) - \lambda_V Var_t + \lambda_S S_t - \lambda_K K_t \, , t = 0, ..., T-1
$$

Clearly, the expected reward at time t is no-longer quadratic in the action a_t, but it is raised to the power four. It corresponds to the portfolio selection problem between time t and $t+1$ in the case of non-normal market rates.
We can then derive the Q-value function and use some of the methods described in Section (9.4) to solve this optimisation problem. However, we can no-longer rely on analytical solutions.

Electronic copy available at: https://ssrn.com/abstract=3307566

13.2 Appendices

13.3 Miscellaneous

13.3.1 The replication portfolio

In a complete market, every contingent claim can be replicated with a trading strategy, or replicating portfolio, made of the underlying asset X and a risk-free money account. For instance, Black and Scholes derived the price of an option by constructing a replicating portfolio (see details in Section (13.1.1.1)). This is not the case in an incomplete market.
We are briefly going to introduce the theory of portfolio replication in complete and incomplete markets (details can be found in Oksendal [1998]).

We let $(X(t))_{t\in[0,T]}$ be a continuous semimartingale on the horizon $[0,T]$, representing the price process of a risky asset. We let a contingent claim be given by $H = h(X(T))$ for some function h, where H is a random variable such that $H \in \mathcal{L}^2(\Omega, \mathcal{F}, \mathbb{P})$. We let $M(t)$ be the risk-free money account, $\alpha(t)$ is the amount of asset held at time t, and $\beta(t)$ is the money account held at time t.

13.3.1.1 Complete market

In a complete market, the value of a portfolio at time t satisfies

$$V(t) = \alpha(t)X(t) + \beta(t)M(t)\,, 0 \leqslant t \leqslant T \tag{13.3.13}$$

For simplicity of exposition we set $M(t) = 1$ for all $0 \leqslant t \leqslant T$. The trading strategy (α, β) is admissible, such that the value process $V(t)$ is square-integrable and have right-continuous paths defined by

$$V(t) = V_0 + \int_0^t \alpha(s)dX(s)$$

For $\mathbb{Q}$-almost surely, every contingent claim H is attainable and admits the representation

$$V(T) = H = V_0 + \int_0^T \alpha(s)dX(s)$$

where $V_0 = E^Q[H]$. The strategy is self-financing, meaning the cost of the portfolio is a constant

$$V(t) - \int_0^t \alpha(s)dX(s) = V_0$$

where V_0 is a perfect hedge. That is, all future changes in the hedge portfolio are funded from an initially set bank account, without any cash injection or withdrawal over the option lifetime.

13.3.1.2 Incomplete market

Exposures to uncertain future events constitutes a basis of arbitrage opportunity, called intrinsic risk. Since the complete market theory failed to account for well-known market anomalies, an incomplete market theory developed to understand and explain such anomalies. For instance, Follmer et al. [1990] derived a replicating portfolio in incomplete market. Letting Π be the set of all intrinsic risks, $G(\pi)$ is a measure of intrinsic risk $\pi(\omega)$, mapping from Π to $\mathbb{R}$. The measure G depends on the underlying asset as well as the contingent claim, such that $G^H \in \mathcal{L}^2(\Omega, \mathcal{F}, \mathbb{P})$, leading to the new representation of the claim H as

Electronic copy available at: https://ssrn.com/abstract=3307566

$$H = V_0 + \int_0^T \alpha(s)dX(s) + G^H \tag{13.3.14}$$

Introducing the Kunita-Watanabe decomposition, the measure can be expressed as

$$G^H = G_0 + \int_0^T \alpha^H(s)dX(s) + N(T)$$

where $N(t)$ is a square-integrable martingale orthogonal to X. Thus, we have

$$H = V_0^* + \int_0^T \alpha^*(s)dX(s) + N(T)$$

where $V_0^* = V_0 + G_0$ and $\alpha^* = \alpha + \alpha^H$. Market incompleteness implies the existence of an equivalent measure in the set $\mathcal{Q}$, which is not necessarily a martingale and/or a unique measure.

13.3.2 Local risk minimisation

We briefly describe a local risk minimisation approach presented by Grau [2007] and modified by Halperin [2017] to obtain an optimal strategy and a pricing formula by minimising portfolio variance as a measure of risk. This method is consistent with the assumption normality for the market returns. We use the same notation as in Appendix (13.3.1).

13.3.2.1 Portfolio evaluation

We consider the seller of an option with maturity T and terminal payoff H. Starting from maturity T, we want to find the amount needed to be held in the bank account at the previous times $t < T$. Given a self-financing strategy, the following relation ensures conservation of the portfolio value by re-hedging at time t:

$$\alpha(t)X(t+1) + e^{r\Delta t}\beta(t) = \alpha(t+1)X(t+1) + \beta(t+1) \tag{13.3.15}$$

We can therefore get a recursive relation used to compute the amount of money to keep in the bank account to hedge the option at any time $t < T$:

$$\beta(t) = e^{-r\Delta t}\Big(\beta(t+1) + \big(\alpha(t+1) - \alpha(t)X(t+1)\big)\Big)\,, t = T-1, ..., 0$$

Combining this with the portfolio value in Equation (13.3.13), we get a recursive relation for the portfolio $V(t)$ in terms of its values at later times, which we solve backward in time with terminal condition H. We get the hedged portfolio

$$V(t) = e^{-r\Delta t}\Big(V(t+1) - \alpha(t)\Delta X(t)\Big)\,, \Delta X(t) = X(t+1) - e^{r\Delta t}X(t)\,, t = T-1, ..., 0 \tag{13.3.16}$$

In that setting, both $\beta(t)$ and $V(t)$ are not measurable at any time $t < T$ since they depend on the future. All uncertainties in $V(t)$ are due to the uncertainty regarding the amount $\beta(t)$ to be held at the bank at time t to cover the payoff H at maturity T. At time $t = 0$, both $\beta(0)$ and $V(0)$ are random quantities with some distributions.
In that setting, the computation of the hedge portfolio value is done path-wise. Given some hedging strategies $\{\alpha(t); t = 0, ..., T\}$ we can infer these distributions by using Monte Carlo simulations. We can simulate N paths for X and go backward on each path to evaluate $V(t)$ (see Grau [2007]). That is,

1. In the forward pass we simulate the process $X(1) \to X(2) \to \cdots \to X(N)$.
2. In the backward pass we use the recursion in Equation (13.3.16) that takes a given hedge strategy $\{\alpha(t); t = 0, ..., T\}$ and back-propagate future uncertainties till today via the self-financing constraint in Equation (13.3.15).

Electronic copy available at: https://ssrn.com/abstract=3307566

We can also estimate $V(0)$ from historical data for X together with a pre-determined hedging strategy and a terminal condition.

13.3.2.2 Optimal hedging

In order to set the price of the option at inception, the seller first need to select a hedging strategy α to be used in the future. In that setting, the theory relies on the principle of hedging first and pricing second. Thus, we need to learn the optimal strategy α^* over all states of the world. That is, optimal hedges must be computed backward in time, starting from $t = T$, and using a cross-sectional analysis on all paths.
Following a similar approach to the pricing of Amrican options (see Longstaff et al. [2001]), Halperin [2017] computed the optimal hedge α_t^* at time t by minimising the portfolio variance across all simulated MC paths, when conditioned on the currently available cross-sectional information $\mathcal{F}_t$. Using Equation (13.3.16) for the recursive portfolio, we get the optimal hedge at time t as

$$\begin{aligned}\alpha_t^* &= \arg\min_\alpha Var_t(V(t)) \\ &= \arg\min_\alpha Var_t\big(V(t+1) - \alpha(t)\Delta X(t)\big)\,, t = T-1, ..., 0\end{aligned}$$

where $Var_t(X) = Var(X|\mathcal{F}_t)$. The optimal hedge should minimise the cost of hedge capital at each time step. Thus, differentiating the above equation and setting the result to zero, we get the optimal hedge at time t as

$$\alpha_t^* = \frac{Cov_t(V(t+1), \Delta X(t))}{Var(\Delta X(t))}\,, t = T-1, ..., 0$$

These terms are just one-step ahead conditional expectations:

- In the case of discrete state space they are finite sums involving the transition probabilities of a MDP model. That is, the finite sums are over all feature discrete states reachable in one step from a given discrete state.
- In the case of continuous state space they can be computed with a Monte Carlo engine by using expansions in basis functions (see Section (9.4.2.4)).

13.3.2.3 The risk-averse price

The price of a fair option at time t is the expected value of the hedged portfolio given by

$$\widehat{C}_t = E[V(t)|\mathcal{F}_t]$$

Using Equation (13.3.16) for the recursive portfolio and conditional expectation, we get

$$\begin{aligned}\widehat{C}_t &= E[e^{-r\Delta t}V(t+1)|\mathcal{F}_t] - \alpha(t)E[\Delta X(t)|\mathcal{F}_t] \\ &= E[e^{-r\Delta t}E[V(t+1)|\mathcal{F}_{t+1}]|\mathcal{F}_t] - \alpha(t)E[\Delta X(t)|\mathcal{F}_t] \\ &= E[e^{-r\Delta t}\widehat{C}_{t+1}|\mathcal{F}_t] - \alpha(t)E[\Delta X(t)|\mathcal{F}_t]\,, t = T-1, ..., 0\end{aligned}$$

Using the same approach, we can express the optimal hedge in terms of $\widehat{C}_{t+1}$ instead of $V(t+1)$, getting

$$\alpha_t^* = \frac{Cov_t(\widehat{C}_{t+1}, \Delta X(t))}{Var(\Delta X(t))}\,, t = T-1, ..., 0$$

Taking the first-order Taylor expansion on the fair price $\widehat{C}$, we get

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\widehat{C}_{t+1} = C_t + \frac{\partial C_t}{\partial X(t)}\Delta X(t) + O(\Delta t)$$

and taking the limit $\Delta t \to 0$, the optimal strategy simplifies to

$$\alpha_t^{BS} = \lim_{\Delta t \to 0} \alpha_t^* = \frac{\partial C_t}{\partial X(t)}$$

which is the BS-delta.
Substituting the optimal hedge in the pricing equation above, we get

$$\widehat{C}_t = e^{-r\Delta t} E^{\widehat{Q}}[V(t)|\mathcal{F}_t]\,, t = T-1, ..., 0$$

where $\widehat{Q}$ is a signed measure with the following transition probabilities:

$$\tilde{q}(X(t+1)|X(t)) = p(X(t+1)|X(t))\Big(1 - \frac{\big(\Delta X(t) - E_t[\Delta X(t)]\big)E_t[\Delta X(t)]}{Var_t(\Delta X(t))}\Big)$$

where $p(X(t+1)|X(t))$ are transition probabilities under the historical measure $\mathbb{P}$. The measure is signed because the transition probabilities can become negative for large moves $\Delta X(t)$. It is a property of quadratic risk minimisation schemes.

***Remark* 13.3.1** *As such this pricing formula does not rely on the no-arbitrage principle. The author argue that if the data respect the no-arbitrage principle, the model will also respect it because it is directly build from data. This is very difficult to prove in practice*

Note, the seller has to compensate for the risk of exhausting the bank account $\beta(t)$ before reaching maturity and having to inject extra cash in the system. Potters et al. [2001] suggested to add the cumulative expected discounted variance of the hedge portfolio along all time steps with a risk-aversion parameter λ. The pricing formula becomes

$$C_0^{ask}(X, \alpha) = E_0[V(0) + \lambda \sum_{t=0}^{T} e^{-rt} Var_t(V(t))|X(0) = X, \alpha_0 = \alpha]$$

***Remark* 13.3.2** *This pricing formula can be made non-negative by choosing a proper level for the risk aversion parameter λ.*

13.3.3 The capital asset pricing model

In the problem of portfolio selection, the mean-variance approach introduced by Markowitz [1952] is a simple trade-off between return and uncertainty, where one is left with the choice of one free parameter, the amount of variance acceptable to the individual investor. For proofs and rigorous introduction to the mean-variance portfolio technique see Huang et al. [1988]. For a retrospective on Markowitz's portfolio selection see Rubinstein [2002]. Investment theory based on growth is an alternative to utility theory with simple goal. Following this approach, Kelly [1956] used the role of time in multiplicative processes to solve the problem of portfolio selection.

13.3.3.1 Mean-variance criterion

13.3.3.1.1 Normal returns We assume that an investor has an exponential utility (EU) function with a Coefficient of Absolute Risk Aversion (CARA) γ given by

$$u(W) = -e^{-\gamma W}\,, \gamma > 0$$

Electronic copy available at: https://ssrn.com/abstract=3307566

and we further assume that the returns on a portfolio are normally distributed with expectation μ and standard deviation σ. Hence, the certain equivalent (CE) of the portfolio can be approximated as

$$CE \approx \mu - \frac{1}{2}\gamma\sigma^2$$

The expected portfolio return is

$$\mu = w^\top E[r]$$

where w is the vector of portfolio weights and r is the vector of returns on the constituent assets, and the portfolio variance is

$$\sigma^2 = w^\top Q w$$

where Q is the covariance matrix of the asset returns. Since the best investment is the one giving the maximum certain equivalent, then for an investor with an exponential utility function investing in risky assets with normally distributed returns, the optimal allocation problem may be approximated by the simple optimisation

$$\max_w \left(\mu - \frac{1}{2}\gamma\sigma^2\right) = \max_w \left(w^\top E[r] - \frac{1}{2}\gamma w^\top Q w\right) \tag{13.3.17}$$

which is the mean-variance criterion. Note, when the utility is defined on the returns on the investment, we must multiply the utility function by the amount invested to find the utility of each investment. Similarly, to find the certain equivalent of a risky investment we multiply by the amount invested. However, we need to express the CRA γ as a proportion of the amount invested.

13.3.3.1.2 Non-normal returns Assuming that investors borrow at zero interest rate, we consider the utility associated with an investment as being defined on the distribution of investment returns rather than on the distribution of the wealth arising from the investment. We further assume that the returns are non-normally distributed. Applying the expectation operator to a Taylor expansion of $u(R)$ about $u(\mu)$, the utility associated with the mean return, we get

$$E[u(R)] = u(\mu) + u^{'}(R)|_{R=\mu}E[R-\mu] + \frac{1}{2}u^{''}(R)|_{R=\mu}E[(R-\mu)^2] + \frac{1}{6}u^{'''}(R)|_{R=\mu}E[(R-\mu)^3] + ...$$

which is a simple approximation to the certain equivalent associated with any utility function since $E[u(R)] = E[u(X)] = u(CE(X))$.
If we assume that the investor has an exponential utility (EU) function, then, the above equation to the fourth order becomes

$$e^{-\gamma CE} \approx e^{-\gamma\mu}\left(1 + \frac{1}{2}\gamma^2 E[(R-\mu)^2] - \frac{1}{6}\gamma^3 E[(R-\mu)^3] + \frac{1}{24}\gamma^4 E[(R-\mu)^4]\right)$$

Given the first four moments are

$$\sigma^2 = E[(R-\mu)^2], S = E[(R-\mu)^3], K = E[(R-\mu)^4]$$

where S is the skew and K is the kurtosis, we get the approximation

$$e^{-\gamma CE} \approx e^{-\gamma\mu}\left(1 + \frac{1}{2}(\gamma\sigma)^2 - \frac{S}{6}(\gamma\sigma)^3 + \frac{K-3}{24}(\gamma\sigma)^4\right)$$

where $K-3$ is the excess kurtosis. Taking the logarithm and using the second order Taylor expansion, the approximated certain equivalent (CE) associated with the exponential utility function simplifies to

Electronic copy available at: https://ssrn.com/abstract=3307566

$$CE \approx \mu - \frac{1}{2}\gamma\sigma^2 + \frac{S}{6}\gamma^2\sigma^3 - \frac{K-3}{24}\gamma^3\sigma^4$$

The mean-variance criterion is a special case of the above equation with no skewness and no kurtosis.

***Remark* 13.3.3** *In general, a risk averse investor having an exponential utility has aversion to risk associated with increasing variance, negative skewness, and increasing kurtosis.*

13.3.3.2 Markowitz solution to the portfolio allocation problem

A rational investor should allocate his funds between the different risky assets in his universe, leading to the portfolio allocation problem. To solve this problem Markowitz [1952] introduced the concept of utility functions to express investor's risk preferences. Markowitz first considered the rule that the investor should maximise discounted expected, or anticipated returns (which is linked to the St. Petersurg paradox). However, he showed that the law of large numbers can not apply to a portfolio of securities since the returns from securities are too intercorrelated. That is, diversification can not eliminate all variance. Hence, rejecting the first hypothesis, he then considered the rule that the investor should consider expected return a desirable thing and variance of return an undesirable thing.

The mean-variance efficient portfolios are obtained as the solution to a quadratic optimisation program. Its theoretical justification requires either a quadratic utility function or some fairly restrictive assumptions on the class of return distribution, such as the assumption of normally distributed returns.
For instance, we assume zero transaction costs and portfolios with prices V_t taking values in $\mathbb{R}$ and following the geometric Brownian motion with dynamics under the historical probability measure $\mathbb{P}$ given by

$$\frac{dV_t}{V_t} = \mu dt + \sigma_V dW_t \tag{13.3.18}$$

where μ is the drift, σ_V is the volatility and W_t is a standard Brownian motion. Markowitz first considered the problem of maximising the expected rate of return

$$g = \frac{1}{dt} < \frac{dV_t}{V_t} >= \mu$$

(also called ensemble average growth rate) and rejected such strategy because the portfolio with maximum expected rate of return is likely to be under-diversified, and as a result, to have an unacceptable high volatility. As a result, he postulated that while diversification would reduce risk, it would not eliminate it, so that an investor should maximise the expected portfolio return μ while minimising portfolio variance of return σ_V^2. It follows from the relation between the variance of the return of the portfolio σ_V^2 and the variance of return of its constituent securities σ_j^2 for $j = 1, 2, ..., N$ given by

$$\sigma_V^2 = \sum_j w_j^2\sigma_j^2 + \sum_j \sum_{k \neq j} w_j w_k \rho_{jk}\sigma_j\sigma_k$$

where the w_j are the portfolio weights such that $\sum_j w_j = 1$, and ρ_{jk} is the correlation of the returns of securities j and k. Therefore, $\rho_{jk}\sigma_j\sigma_k$ is the covariance of their returns. So, the decision to hold any security would depend on what other securities the investor wants to hold. That is, securities can not be properly evaluated in isolation, but only as a group. Consequently, Markowitz suggested calling portfolio i efficient if

1. there exists no other portfolio j in the market with equal or smaller volatility, $\sigma_j \leqslant \sigma_i$, whose drift term μ_j exceeds that of portfolio i. That is, for all j such that $\sigma_j \leqslant \sigma_i$, we have $\mu_j \leqslant \mu_i$.
2. there exists no other portfolio j in the market with equal or greater drift term, $\mu_j \geqslant \mu_i$, whose volatility σ_j is smaller than that of portfolio i. That is, for all j such that $\mu_j \geqslant \mu_i$, we have $\sigma_j \geqslant \sigma_i$.

Electronic copy available at: https://ssrn.com/abstract=3307566

In the presence of a riskless asset (with $\sigma_i = 0$), all efficient portfolios lie along a straight line, the efficient frontier, intersecting in the space of volatility and drift terms, the riskless asset r_f and the so-called market portfolio M. Since any point along the efficient frontier represents an efficient portfolio, additional information is needed in order to select the optimal portfolio. For instance, one can specify the usefulness or desirability of a particular investment outcome to a particular investor, namely his risk preference, and represent it with a utility function $u = u(V_t)$. Following the work of von Neumann et al. [1944] and Savage [1954], Markowitz [1959] found a way to reconcile his mean-variance criterion with the maximisation of the expected utility of wealth after many reinvestment periods. He advised using the strategy of maximising the expected logarithmic utility of return each period for investors with a long-term horizon, and developed a quadratic approximation to this strategy allowing the investor to choose portfolios based on mean and variance.

13.3.3.3 The expected growth rate

As an alternative to the problem of portfolio selection, Kelly [1956] proposed to maximise the expected growth rate

$$g^b = \frac{1}{dt} < d\ln V_t >= \mu - \frac{1}{2}\sigma^2$$

obtained by using Ito's formula (see Peters [2011c]). This rate is called the expected growth rate, or the logarithmic geometric mean rate of return (also called the time average growth rate). In that setting, we observe that large returns and small volatilities are desirable. Note, Ito's formula changes the behaviour in time without changing the noise term. That is, Ito's formula encode the multiplicative effect of time (for noise terms) in the ensemble average (see Oksendal [1998]). Hence, it can be seen as a mean of accounting for the effects of time. For self-financing portfolios, where eventual outcomes are the product over intermediate returns, maximising g^b yields meaningful results. This is because it is equivalent to using logarithmic utility function $u(V_t) = \ln V_t$. In that setting, the rate of change of the ensemble average utility happens to be the time average of the growth rate in a multiplicative process. Note, the problem that additional information is needed to select the right portfolio disappears when using the expected growth rate g^b. That is, there is no need to use utility function to express risk preferences as one can use solely the role of time in multiplicative processes.

13.3.4 Point on Kolmogorov-Compatibility

Dealing with deterministic smile surface evolution always rises the question of whether the smile surface allows for arbitrage opportunities (see Section (13.1.1.4)). It is natural to impose that future smile surface are compatible with today's prices of calls and puts. The problem is that defining a deterministic smile surface $\sigma(t, T, K, S_t)$ means defining a future deterministic density ϕ for stock process, and then a natural condition would be to impose that the future pdf is actually a conditional density. This is what we will formalise as the Kolmogorov-Compatibility. First of all, let's remind the relation between prices and pdf. Without doing any assumptions apart from that of no arbitrage opportunity, we have

$$\begin{aligned}
C(t, K, t_1 - t_0) &= e^{-r(t_1-t_0)}\left((S_{t_1} - K)^+ | \mathcal{F}_{t_1}\right) = e^{-r(t_1-t_0)}\int_K^{+\infty}(x_1 - K)\phi(x_1, t_1, S_{t_0}, t_0)d\lambda(x_1) \\
\frac{\partial C}{\partial K}(t, K, t_1 - t_0) &= -e^{-r(t_1-t_0)}\int_K^{+\infty}\phi(x_1, t_1, S_{t_0}, t_0)d\lambda(x_1) \\
\frac{\partial^2 C}{\partial K^2}(t, K, t_1 - t_0) &= e^{-r(t_1-t_0)}\phi(K, t_1, S_{t_0}, t_0)
\end{aligned}$$

By definition $C(t, K, t_1 - t_0) = C_{BS}(S_{t_0}; K, t_1 - t_0; r, \Sigma(S_{t_0}; K, t_1 - t_0))$, such that applying the chain rule we have

$$\frac{\partial^2 C}{\partial K^2} = \frac{\partial^2 C_{BS}}{\partial K^2} + \frac{\partial^2 C_{BS}}{\partial K \partial \Sigma}\frac{\partial \Sigma}{\partial K} + \left[\frac{\partial^2 C_{BS}}{\partial K \partial \Sigma} + \frac{\partial^2 C_{BS}}{\partial K^2}\frac{\partial \Sigma}{\partial K}\right]\frac{\partial \Sigma}{\partial K} + \frac{\partial C_{BS}}{\partial \Sigma}\frac{\partial^2 \Sigma}{\partial K^2}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

For the sake of clarity we denote by Θ the right-hand-side operator.

Definition 13.3.1 (Kolmogorov-Compatibility) *Any future deterministic conditional density, or smile surface, such that*

$$\Theta(S_0; K, T-t_0; r, \sigma(t_0, T, K, S_{t_0})) = \int \Theta(S_0; K, t_1 - t_0; r, \sigma(t_0, t_1, K, S_{t_0}))\Theta(S_{t_1}; K, T-t_1; r, \sigma(t_1, T, K, S_{t_1}))d\lambda(S_{t_1})$$

is satisfied, defines a Kolmogorov-compatible density.

Property 13.3.1 *Given a current admissible*[4] *smile surface, if the future smile surface is Kolmogorov-compatible no model-independent strategy can generate arbitrage profits.*

There is in general an infinity of solutions for the forward density

$$\Theta(S_{t_1}; K, T-t_1; r, \sigma(t_1, T, K, S_{t_1}))$$

such that Definition (13.3.1) is satisfied. Therefore, even if we require the smile surface to be deterministic, there still exists an infinity of future smile surfaces compatible with today's prices of calls and puts. Additional information is needed to determine these conditional distributions. We need to know all forward smiles, that is, the future prices of all vanilla options as seen from all possible states of the world.

13.4 The parametric MixVol model

Bloch [2010] [2012] considered a parametric interpolation and extrapolation of the implied volatility (IV) surface by decomposing the market option prices into a weighted sum of strike shifted Black-Scholes counterparts. A term structure of volatility was introduced to infer plausible deformation of the IV surface both in space and in time. A Differential Evolution algorithm was used to solve a non-linear optimisation problem under constraints. We are now going to describe that model.

13.4.1 Description of the model

To obtain a pronounced skew we consider a sum of shifted log-normal distributions, that is, using the Black-Scholes formula with shifted strike (modified by the parameters $\mu_i(t)$) as an interpolation function. In our parametric model, the market option price $C_M(K,t)$, for a strike K and maturity t, is estimated at time $t_0 = 0$ by the weighted sum

$$C_M(t_0, S_0, P_t, R_t, D_t; K, t) = \sum_{i=1}^{n} a_i(t) Call_{BS}(t_0, S_0, R_t, P_t, \overline{K}(K,t), t, \Sigma_i(t)) \tag{13.4.19}$$

where $a_i(t)$ for $i = 1, .., n$ are the weights, $\overline{K}(K,t) = K^{'}(K,t)(1+\mu_i(t))$ with $K^{'}(K,t) = K + D_t$. In that setting $R_t = Re(0,t)$ is the repo factor in the range $[0,t]$, $P_t = P(0,t) = e^{-rt}$ is the zero-coupon bond price, $C_t = C(0,t) = \frac{R_t}{P_t}$ is the cost of carry and $D_t = D(0,t)$ is the compounded sum of discrete dividends between $[0,t]$. We let the time function $t \to \Sigma_i(t)$ be regular enough, and choose to model directly the square-root of the average variance defined as

$$\Sigma_i^2(t) = \frac{1}{t}\int_0^t \sigma^2(s)ds$$

[4] *i.e.* such that the associated call prices sarisfy $\frac{\partial Call(t,K,T-t)}{\partial K} < 0$, $\frac{\partial^2 Call(t,K,T-t)}{\partial K^2} > 0$, $\frac{\partial Call(t,K,T-t)}{\partial T} > 0$, $\frac{\partial Put(t,K,T-t)}{\partial K} > 0$, $Call(t,K,T-t)|_{K=0} = S_t$ and $lim_{K\to\infty} Call(t,K,T-t) = 0$

Electronic copy available at: https://ssrn.com/abstract=3307566

where $\sigma(t)$ is the instantaneous volatility of the underlying process. To guarantee the positivity of the local volatility, the average variance must verify

$$\Sigma_i^2(t) + 2\Sigma_i(t)t\partial_t\Sigma_i(t) \geqslant 0 \ , \forall i$$

and since $\Sigma_i(t) > 0$, the constraint simplifies to

$$\Sigma_i(t) \geqslant -2t\partial_t\Sigma_i(t) \ , \forall i \tag{13.4.20}$$

Further, the no-arbitrage theory imposes time and space constraints on market prices. Introducing the time dependent parameters $a_i(t)$ and $\mu_i(t)$, the simplest way of ensuring these constraints is to take the same time dependency for each μ, that is, $\mu_i(t) = \mu_i f(t, \beta_i)$. As the pdf of the equity price should tend toward a single Dirac when $t \to 0$, to get control on $\mu_i(t)$, we choose to let the function $f(t, x)$ tend to 1 when $t \to +\infty$, getting

$$f(t,x) = 1 - \frac{2}{1 + (1 + \frac{t}{x})^2}$$

with $f^{'}(t,x) = [1 - f(t,x)]^2 \frac{1}{x}(1 + \frac{t}{x})$. Moreover, to keep manageable the no-free lunch constraints, we make the weight $a_i(t)$ proportional to $\frac{a_i^0}{f(t,\beta_i)}$ for some constant $a_i^0 > 0$, getting the representation

$$\mu_i(t) = \mu_i^0 f(t, \beta_i) \text{ and } a_i(t) = \frac{a_i^0}{f(t,\beta_i) \times norm}$$

where $norm = \sum_{i=1}^n \frac{a_i^0}{f(t,\beta_i)}$. As a result, with seperable functions of time, we can prove that the no-free lunch constraints simplify.

Theorem 13.4.1 *No-Arbitrage Constraints in the MixVol Model*
We let the price of a European call option be given by Equation (13.4.19) and assume seperable functions of time $a_i(t)$, $\mu_i(t)$. Then the resulting implied volatility surface is free from static arbitrage (according to Definition (13.1.4)) provided the following parameter restrictions hold:

$$\begin{aligned} a_i^0 &\geqslant 0 \\ \sum_{i=1}^n a_i(t) &= 1 \\ \sum_{i=1}^n a_i^0 \mu_i^0 &= 0 \\ \mu_i^0 &\geqslant -1 \end{aligned} \tag{13.4.21}$$

To make sure that the IV surface flattens for infinitely large expiries, we impose the limit behaviour

$$\lim_{t\to\infty} \Sigma_i(t) = d_i \tag{13.4.22}$$

where $d_i > 0$. For example, Bloch [2010] discussed the volatility function

$$\Sigma_i(t) = \big(a_i + b_i t\big)e^{-c_i t} + d_i$$

where one must impose $b_i \geqslant 0$ for $\Sigma_i(.)$ to remain positive, generating only upward humps.
To get a general volatility function capable of generating both an upward hump or a downward one, Bloch [2012] proposed the volatility function

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\Sigma_i(t) = \big(a_i + b_i \ln(1 + e_i t)\big)e^{-c_i t} + d_i$$

where $c_i > 0$, $d_i > 0$ and $a_i \in \mathbb{R}$, $b_i \in \mathbb{R}$ and $e_i \in]-\frac{1}{t}, \infty[$. The derivative of the function with respect to time t is

$$\Sigma_i^{'}(t) = \big(-a_i c_i + b_i \frac{e_i}{1 + e_i t} - b_i c_i \ln(1 + e_i t)\big)e^{-c_i t}$$

and the no-arbitrage constraint must satisfy

$$e^{-c_i t}\big[(1 - 2tc_i)\big(a_i + b_i \ln(1 + e_i t)\big) + 2tb_i \frac{e_i}{1 + e_i t}\big] + d_i \geqslant 0$$

Since $a_i \in \mathbb{R}$, at time $t = 0$ the left hand side of the inequality can becomes negative. Consequently, we must impose the constraint

$$a_i + d_i > 0$$

to get the constraint satisfied at $t = 0$. Further, when $t > \frac{1}{c_i}$ then for c_i sufficiently large the constant d_i will dominates $\big(a_i + b_i \ln(1 + e_i t)\big)$ ensuring positivity of the left hand side. Hence, at time $t = \frac{1}{c_i}$ we must impose

$$d_i \geqslant \big(a_i - 2\frac{b_i}{c_i}\frac{e_i}{1 + \frac{e_i}{c_i}} + b_i \ln(1 + \frac{e_i}{c_i})\big)e^{-1}$$

13.4.2 Computing the Greeks

Assuming that the dynamics of the spot price with discrete dividends follow the Spot model, our parametric model in Equation (13.4.19) corresponds to a weighted sum of Black-Scholes formulas in the Z-space. In that setting, the derivative of a call option price with respect to the spot is

$$\frac{\partial}{\partial S_0} C_M(t_0, S_0, P_t, R_t, D_t; K, t) = \frac{1}{norm} \sum_{i=1}^{n} \overline{a}_i(t) \frac{\partial}{\partial S_0} Call_{BS}(t_0, S_0, R_t, 1, \tilde{K}(K, t), t, \Sigma_i(t))$$

where $\tilde{K}(K, t) = P(t_0, t)\overline{K}(K, t)$ and $\overline{K}(K, t) = K^{'}(K, t)(1 + \mu_i(t))$ and $\overline{a}_i(t) = \frac{a_i^0}{f(t, \beta_i)}$. Differentiating one more time the call price with respect to the spot, we get

$$\frac{\partial^2}{\partial S_0^2} C_M(t_0, S_0, P_t, R_t, D_t; K, t) = \frac{1}{norm} \sum_{i=1}^{n} \overline{a}_i(t) \frac{\partial^2}{\partial S_0^2} Call_{BS}(t_0, S_0, R_t, 1, \tilde{K}(K, t), t, \Sigma_i(t))$$

Differentiating the call price with respect to the strike, we get

$$\frac{\partial}{\partial K} C_M(t_0, S_0, P_t, R_t, D_t; K, t) = \frac{1}{norm} \sum_{i=1}^{n} \overline{a}_i(t) \frac{\partial}{\partial \tilde{K}} Call_{BS}(t_0, S_0, R_t, 1, \tilde{K}(K, t), t, \Sigma_i(t)) \frac{\partial \tilde{K}}{\partial K} \tag{13.4.23}$$

where $\frac{\partial \tilde{K}}{\partial K} = P(t_0, t)(1 + \mu_i(t))$.

13.4.3 Scenarios analysis

We are going to present a few stress scenarios defined on the IV surface to visualise and quantify the risk inherent to option portfolios. We choose to consider a range of plausible scenarios for the dynamics of the implied volatility surface. To do so we will modify the model parameters such that the the no-arbitrage constraints are satisfied. In order to obtain an upward or downward movement of the skew and curvature on the IVS, we are going to bump each

Electronic copy available at: https://ssrn.com/abstract=3307566

function's curve $f_i(t)$ inside the shift function $\mu_i(t)$ with a constant. As a result, one of the shifted parametric price becomes

$$C_M(t_0, S_0, P_t, R_t, D_t; K, t; \epsilon) = \frac{1}{norm} \sum_{i=1}^{n} \overline{a}_i(t) Call_{BS}(t_0, S_0, R_t, P_t, K^{'}(K,t)(1 + \mu_i^0(f_i(t) + \epsilon)), t, \Sigma_i(t))$$

Rather than modifying the function $f_i(t)$, we can directly modify the parameter β_i in the previous parametric price $C_M(t_0, S_0; K, t)$, getting $f(t, \beta_i(1+\epsilon))$ with $\epsilon << 1$. In addition, multiplying each parameter μ_i^0 with the same constant $(1+\epsilon)$, rather than adding a constant, will preserve the no-arbitrage constraints. Hence, under that scenario, the shifted option price becomes

$$C_M(t_0, S_0, P_t, R_t, D_t; K, t; \epsilon) = \frac{1}{norm} \sum_{i=1}^{n} \overline{a}_i(t) Call_{BS}(t_0, S_0, R_t, P_t, K^{'}(K,t)(1 + (1+\epsilon)\mu_i^0 f_i(t)), t, \Sigma_i(t))$$

where $\epsilon << 1$. Alternatively, we can multiply each parameter μ_i^0 with the constant $(1+\epsilon_i)$ if we force one degree of liberty as

$$\epsilon_1 = -\frac{1}{a_1^0 \mu_1^0} \sum_{i=2}^{n} a_i^0 \mu_i^0 (1 + \epsilon_i)$$

One can also bump each μ_i^0 with ϵ_2 for $i = 1, .., n$, but in order to satisfy the no-arbitrage constraint we also need to bump each weight a_i^0 with ϵ_1 such that

$$\epsilon_1 = -\frac{\epsilon_2}{\sum_{i=1}^{n} \mu_i^0 + n\epsilon_2}$$

As the representation of the IVS is a highly non-linear function of the term structures $\Sigma_i(t)$ as well as the shift parameters $\mu_i(t)$, adding a positive constant ϵ to each term structure $\Sigma_i(t)$ will produce a parallel shift in the MixVol term structure $\widehat{\sigma} = \sum_{i=1}^{n} a_i(t)\Sigma_i(t)$ but not in the IVS itself. That is, for $t_1 > t_0$ the movement of the new implied volatility surface $\Sigma(t_1; K^{'}, t)$ produced by that scenario depends on the shape of the IVS at inception $\Sigma(t_0; K^{'}, t)$ which is in accordance with the result found by Rogers et al. [2010]. The smaller the skew, the larger the impact of $\widehat{\sigma}$ on the IV surface. As a result, we can shift each BS function $\Sigma_i(t)$ with a positive constant to generate a parallel shift type of movements of the IVS around the money, getting the shifted parametric price

$$C_M(t_0, S_0, P_t, R_t, D_t; K, t; \epsilon) = \frac{1}{norm} \sum_{i=1}^{n} \overline{a}_i(t) Call_{BS}(t_0, S_0, R_t, P_t, \overline{K}(K,t), t, \Sigma_i(t) + \epsilon)$$

where $\epsilon = 1\%$ to be consistent with the Black-Scholes vega. We can also define another type of volatility scenario dependent on the maturity with

$$(1+\epsilon)\Sigma_i^2(t) = (1+\epsilon)\frac{1}{t}\int_0^t \sigma^2(s)ds$$

where $\epsilon << 1$, so that the shifted option price becomes

$$C_M(t_0, S_0, P_t, R_t, D_t; K, t; \epsilon) = \frac{1}{norm} \sum_{i=1}^{n} \overline{a}_i(t) Call_{BS}(t_0, S_0, R_t, P_t, \overline{K}(K,t), t, \Sigma_i(t)(1+\epsilon))$$

Combining all these bumps together, we obtain a no-arbitrage deformation of the IV surface generated by the shifted option price

Electronic copy available at: https://ssrn.com/abstract=3307566

$$
\begin{aligned}
& C_M(t_0, S_0, P_t, R_t, D_t; K, t; \epsilon_\mu, \epsilon_\beta, \epsilon_\sigma) \\
= & \frac{1}{norm} \sum_{i=1}^{n} \overline{a}_i(t) Call_{BS}(t_0, S_0, R_t, P_t, K^{'}(K,t)(1 + (1+\epsilon_\mu)\mu_i^0 f(t, \beta_i(1+\epsilon_\beta))), t, \Sigma_i(t) + \epsilon_\sigma)
\end{aligned}
$$

For risk management purposes, we can therefore compute the modified vega as

$$\widehat{V}ega = C_M(t_0, S_0, P_t, R_t, D_t; K, t; \epsilon_\mu, \epsilon_\beta, \epsilon_\sigma) - C_M(t_0, S_0, P_t, R_t, D_t; K, t)$$

Electronic copy available at: https://ssrn.com/abstract=3307566

Part VI

Annexes

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 14

Some mathematical facts

14.1 Notation

We let $\mathbb{R}^n$ be the n-dimensional Euclidean space, $\mathbb{R}_+$ be the non-negative real numbers, $\mathbb{Q}$ the rational numbers, $\mathbb{Z}$ the integers, $\mathbb{Z}_+ = \mathbb{N}$ the natural numbers, $\mathbb{C}$ the complex plane, and $R^{n\times m}$ the $n \times m$ matrices with real entries.
We let $C(U, V)$ be the continuous functions from U to V, and $C(U)$ be the same as $C(U, \mathbb{R})$. We denote $C_b(U)$ the bounded continuous functions on U, and $C_0(U)$ or $C_c(U)$ be the functions in $C(U)$ with compact support. Then, $C^n = C^n(U)$ are the functions in $C(U, \mathbb{R})$ with continuous derivatives up to order n. Thus, $C_0^n = C_0^n(U)$ are the functions in C^n with compact support in U. We denote $C^{n+\alpha}$ the functions in C^n whose nth derivatives are Lipschitz continuous with exponent α. At last, $C^{1,2}(\mathbb{R} \times \mathbb{R}^n)$ are the functions $f(t, x) : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ which are C^1 with respect to $t \in \mathbb{R}$ and C^2 with respect to $x \in \mathbb{R}^n$.

Motivated by the algebra of set , where the symbols $\cap$ and $\cup$ represent intersection and union, respectively, these symbols have been introduced in lattice theory where we have join and meet. The least upper bound of $a, b \in A$ is denoted by $a \cup b$ and called the join of elements a, b and the greatest lower bound of $a, b \in A$ is denoted by $a \cap b$ and called the meet of a, b.
In propositional calculus we have $\vee$ for disjunction (introduced by Russell) and we have $\wedge$ for conjunction.
The link between the two notations is with boolean algebra and its use as interpretation for the propositional calculus.
Given the binary relation $\leqslant$ in every Boolean algebra, we write $p \leqslant q$ in case $p \wedge q = p$, and we have that: for each p and q, the set $\{p, q\}$ has the supremum $p \vee q$ and the infimum $p \wedge q$.
The supremum is the minimum of the set of upper bounds, and the infimum the maximum of the set of lower bound.

14.2 Some functions

14.2.1 Miscellaneous

14.2.1.1 The derivative

The essence of differentiation is finding the ratio between the difference in value of $f(x)$ and the increment in x. The derivative, or slope of a function is given by

$$f^{'}(x) = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Generally, if $f = f(g(x))$ is a differentiable function of g and g is a differentiable function of x, then

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}$$

where df, dg, and dx are, respectively, the differentials of the functions f, g and x. This is the chain rule of differentiation which is formally stated as follow:

Theorem 14.2.1 *The chain rule of differentiation*
If g is differentiable in x_0 and if h is differentiable in $u_0 = g(x_0)$, then $h(x) = h(g(x))$ is differentiable in x_0 and

$$\frac{dh}{dx}|_{x=x_0} = \frac{dh}{du}|_{u_0=g(x_0)} \times \frac{dg}{dx}|_{x=x_0}$$

The product rule says that if both $u = u(x)$ and $v = v(x)$ are differentiable functions of x, then

$$\frac{d(uv)}{dx} = v\frac{du}{dx} + u\frac{dv}{dx}$$

Multiply both sides by the differential dx, we obtain

$$d(uv) = vdu + udv$$

which is the product rule in terms of differentials.
For any differentiable function $F(x)$, $dF = F^{'}(x)dx$ (with $F^{'}(x) = \frac{dF}{dx}$). Note, $F(x)$ is one antiderivative of the function $f(x)$ means $F^{'}(x) = f(x)$.

Definition 14.2.1 *Let f be a function defined on some interval (c, d), and let a be some number in this interval. The derivative of f at a is the value of the limit*

$$f^{'}(a) = \lim_{x \to a} = \frac{f(x) - f(a)}{x - a}$$

f is said to be differentiable at a if the limit exists.

The function f is called differentiable on the interval (c, d) if it is differentiable at every point a in (c, d).

Theorem 14.2.2 *If a function f is differentiable at some a in its domain, then f is also continuous at a.*

Definition 14.2.2 *A function F is called an antiderivative of f on the interval $[a, b]$ if one has $F^{'}(x) = f(x)$ for all x with $a < x < b$.*

14.2.1.2 The subgradient

The subderivative, subgradient, and subdifferential generalise the derivative to convex functions which are not necessarily differentiable. Subderivatives arise in convex analysis, the study of convex functions, often in connection to convex optimisation (see Clarke [1983]).

Definition 14.2.3 *Subderivative*
A subderivative of a convex function $f : I \to \mathbb{R}$ at a point x_0 in the open interval I is a real number c such that

$$f(x) - f(x_0) \geqslant c(x - x_0)$$

for all x in I.

Electronic copy available at: https://ssrn.com/abstract=3307566

One can show that the set of subderivatives at x_0 for a convex function is a nonempty closed interval $[a, b]$, where a and b are the one-sided limits

$$\begin{aligned} a &= \lim_{x \to x_0^-} \frac{f(x) - f(x_0)}{x - x_0} \\ b &= \lim_{x \to x_0^+} \frac{f(x) - f(x_0)}{x - x_0} \end{aligned}$$

which are guaranteed to exist and satisfy $a \leqslant b$. The set $[a, b]$ of all subderivatives is called the subdifferential of the function f at x_0. Since f is convex, if its subdifferential at x_0 contains exactly one subderivative, then f is differentiable at x_0.
The concepts of subderivative and subdifferential can be generalised to functions of several variables.

Definition 14.2.4 *Subgradient*
If $f : U \to \mathbb{R}$ is a real-valued convex function defined on a convex open set in the Euclidean space $\mathbb{R}^n$, a vector v in that space is called a subgradient at a point x_0 in U if for any x in U one has

$$f(x) - f(x_0) \geqslant v \cdot (x - x_0)$$

The set of all subgradients at x_0 is called the subdifferential at x_0 and is denoted $\partial f(x_0)$. The subdifferential is always a nonempty convex compact set.
$\partial f(x_t)$ denotes a subgradient of f at x_t and x_t is the t-th iterate of x. If f is differentiable, then its only subgradient is the gradient vector ∇f itself.
These concepts generalize further to convex functions $f : U \to \mathbb{R}$ on a convex set in a locally convex space V. A functional v^* in the dual space V^* is called subgradient at x_0 in U if for any x in U

$$f(x) - f(x_0) \geqslant v^*(x - x_0)$$

14.2.1.3 The kernel functions

Suppose that we would like to learn a nonlinear classification rule which corresponds to a linear classification rule for the transformed data points $\phi(x_i)$. However, when the dimension of $\phi(\cdot)$ becomes very large we should not use an explicit transformation. Note, if $\phi(x)$ is all ordered monomials of degree d, then $\phi^\top(x)\phi(x^{'}) = (x^\top x^{'})^d$ and we can implicitly work in a higher dimensional space by using a different dot product. This is called the kernel trick and

$$\phi^\top(x)\phi(x^{'}) = k(x, x^{'})$$

is a kernel function. Note, we can do an implicit transformation into infinite dimensional feature spaces if we use the right kernel function. For example, the Gaussian kernel function (or radial basis function) is given by

$$k(x, x^{'}) = e^{-\frac{\|x - x^{'}\|_2^2}{2h^2}}$$

We might also want to know the types of kernel functions corresponding to inner products in higher dimensional spaces.

Theorem 14.2.3 *Given some input space X, in order for a kernel function $k : X \to X$ to correspond to an inner product, it must satisfy:*

1. $k(x, x^{'}) = k(x^{'}, x)$, $\forall x, x^{'} \in X$ *(Symmetry)*

Electronic copy available at: https://ssrn.com/abstract=3307566

2. *For $\{x_1, x_2, ..., x_m\} \subset X$, the matrix*

$$K = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_m) \\ \vdots & \ddots & \vdots \\ k(x_1, x_1) & \cdots & k(x_1, x_m) \end{bmatrix}$$

known as the Gram matrix (or kernel matrix) is positive semidefinite.

The second condition is equivalent to each of the following conditions:

- $\forall \alpha \in \mathbb{R}^m$, $\alpha^\top K \alpha \geqslant 0$.
- All the eigenvalues of K are non-negative.

Some of the properties of kernel functions are:

- Suppose k_1, k_2 are kernel functions, $\alpha, \beta \geqslant 0$. Then,

$$k(x, x^{'}) = \alpha k_1(x, x^{'}) + \beta k_2(x, x^{'})$$

  is also a kernel function.

- In particular, if $\alpha = \beta = 1$, we have that

$$k(x, x^{'}) = k_1(x, x^{'}) + k_2(x, x^{'})$$

  is a kernel function.

Some useful kernel functions are the polynomial and radial basis (Gaussian) kernel functions given as follows:

$$\begin{aligned} &\text{Polynomial (inhomogeneous) function } : k(x_i, x_j) = (x_i \cdot x_j + 1)^d \\ &\text{Polynomial (homogeneous) function } : k(x_i, x_j) = (x_i \cdot x_j)^d \\ &\text{Gaussian radial basis function } : k(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2} \\ &\text{Hyperbolic tangent function } : k(x_i, x_j) = \tanh(\kappa x_i \cdot x_j + c) \text{ for some } \kappa > 0 \text{ and } c < 0. \end{aligned}$$

where d is the degree of the polynomial function and $\gamma > 0$ is the constant of the radial basis function. The latter is sometime given by $\gamma = \frac{1}{2}\sigma^2$.

14.2.1.4 The Dirac function

The Dirac delta function, or δ function, is a generalised function, or distribution that was historically introduced by the physicist Paul Dirac for modelling the density of an idealized point mass or point charge, as a function that is equal to zero everywhere except for zero and whose integral over the entire real line is equal to one. Laurent Schwartz introduced the theory of distribution, formalising and validating mathematically these computations. This amounts, in particular, to defining Dirac delta function as a linear functional, which maps every function to its value at zero.
The derivative of the Heaviside function $\mathcal{H}(x)$ is zero for $x \neq 0$, and at $x = 0$ the derivative is undefined. We represent the derivative of the Heaviside function by the Dirac delta function $\delta(x)$. The delta function is zero for $x \neq 0$ and infinite at the point $x = 0$. Since the derivative of $\mathcal{H}(x)$ is undefined, $\delta(x)$ is not a function in the conventional sense of the word. While we can derive the properties of the delta function rigorously, we will focus on heuristic treatments.

Electronic copy available at: https://ssrn.com/abstract=3307566

The Dirac delta function can be loosely thought of as a function on the real line which is zero everywhere except at the origin, where it is infinite. It can be written as follow:

$$\delta(x) = \begin{cases} 0 \text{ for } x \neq 0 \\ \infty \text{ for } x = 0 \end{cases}$$

and

$$\int_{-\infty}^{\infty} \delta(x)dx = 1 \tag{14.2.1}$$

The second property comes from the fact that $\delta(x)$ represents the derivative of $\mathcal{H}(x)$.
The Kronecker delta (named after Leopold Kronecker) is a function of two variables, usually just positive integers. The function is 1 if the variables are equal, and 0 otherwise:

$$\delta_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}$$

where the Kronecker delta δ_{ij} is a piecewise function of variables i and j. It appears naturally in many areas of mathematics, physics and engineering, as a means of compactly expressing its definition above. The following equations are satisfied:

- $\sum_j \delta_{ij} a_j = a_i$
- $\sum_i \delta_{ij} a_i = a_j$
- $\sum_k \delta_{ik}\delta_{kj} = \delta_{ij}$

so that the matrix δ can be considered as an identity matrix. The Kronecker delta has the so-called sifting property that for $j \in \mathbb{Z}$

$$\sum_{i=-\infty}^{\infty} \delta_{ij} a_i = a_j$$

and if the integers are viewed as a measure space, endowed with the counting measure, then this property coincides with the defining property of the Dirac delta function

$$\int_{-\infty}^{\infty} \delta(y - x) f(y) dy = f(x)$$

and in fact Dirac's delta was named after the Kronecker delta because of this analogous property.

14.3 Some facts on convex and concave analysis

Details can be found on text books on convex analysis (see Rockafellar [1995]) as well as on textbooks on mathematics for economics (see Wilson [2012]). We use $\text{relint}(S)$ to refer to the relative interior of a convex set S, which is the set S minus all of the points on the relative boundary. We use $\text{closure}(S)$ to refer to the closure of S, the smallest closed set containing all of the limit points of S. We use ∇ to denote a differential operator that indicates taking gradient in vector calculus. In the Cartesian coordinate system $\mathbb{R}^n$ with coordinates $(x_1, ..., x_n)$ and standard basis $(\hat{e}_1, .., \hat{e}_n)$, the gradient is defined in terms of partial derivative operators as

$$\nabla = (\frac{\partial}{\partial x_1}, .., \frac{\partial}{\partial x_n}) = \sum_{i=1}^{n} \hat{e}_i \frac{\partial}{\partial x_i}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

The gradient product rule is given by

$$\nabla(fg) = f\nabla g + g\nabla f$$

and the rules for dot products satisfy

$$\nabla(u.v) = (u.\nabla)v + (v.\nabla)u + u \times (\nabla \times v) + v \times (\nabla \times u)$$

where u and v are vectors. The directional derivative of a scalar field $f(x, y, z)$ in the direction $a(x, y, z) = a_x\widehat{x} + a_y\widehat{y} + a_z\widehat{z}$ is defined as

$$a \cdot \text{ grad } f = a_x\frac{\partial f}{\partial x} + a_y\frac{\partial f}{\partial y} + a_z\frac{\partial f}{\partial z} = (a \cdot \nabla)f$$

which gives the change of a field f in the direction of a. The Laplace operator is a scalar operator that can be applied to either vector or scalar fields; for Cartesian coordinate systems it is defined as

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} = \nabla.\nabla = \nabla^2 \tag{14.3.2}$$

The tensor derivative of a vector field v can be denoted simply as $\nabla \otimes v$ where $\otimes$ represents the dyadic product. This quantity is equivalent to the transpose of the Jacobian matrix of the vector field with respect to space.

14.3.1 Convex functions

If $x, y \in \mathbb{R}$ and $\alpha \in (0, 1)$, then $(1-\alpha)x+\alpha y$ is a convex combination of x and y. Geometrically, a convex combination of x and y is a point somewhere between x and y. A set $X \subset \mathbb{R}$ is convex if $x, y \in X$ implies $(1 - \alpha)x + \alpha y \in X$ for all $\alpha \in [0, 1]$.

Definition 14.3.1 *A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if any of the following conditions hold*

1. *for all $x, y \in \mathbb{R}^n$,*
$$\frac{1}{2}f(x) + \frac{1}{2}f(y) \geqslant f(\frac{x+y}{2})$$
2. *for all $x, y \in \mathbb{R}^n$ and $\alpha \in [0, 1]$,*
$$(1 - \alpha)f(x) + \alpha f(y) \geqslant f((1 - \alpha)x + \alpha y)$$
3. *for all random variables X, Jensen's inequality*
$$E[f(X)] \geqslant f(E[X]) \tag{14.3.3}$$
is satisfied. If f is strictly convex then the equality implies that $X = E[X]$ w.p. 1.

***Proposition* 1** *A sufficient condition for f to be convex is that*

$$\nabla^2 f \geq 0$$

where ≥ 0 stands for positive semi-definitness, that is, $\nabla^2 f$ has non-negative eigenvalues.

***Proposition* 2** *If f is convex and differentiable, then for all $x_0, \delta \in \mathbb{R}^n$,*

$$f(x_0 + \delta) \geqslant f(x_0) + \delta.\nabla f(x_0)$$

Theorem 14.3.1 *If f has a second derivative which is non-negative (positive) everywhere, then f is convex (strictly convex).*

Electronic copy available at: https://ssrn.com/abstract=3307566

14.3.2 Concave functions

Definition 14.3.2 *A function $f : X \to \mathbb{R}$ is concave if for all $x, y \in X$ and $\alpha \in [0, 1]$,*

$$(1 - \alpha)f(x) + \alpha f(y) \leqslant f((1 - \alpha)x + \alpha y)$$

Definition 14.3.3 *A function $f : X \to \mathbb{R}$ is strictly concave if for all $x, y \in X$ with $x \neq y$ and $\alpha \in [0, 1]$,*

$$(1 - \alpha)f(x) + \alpha f(y) < f((1 - \alpha)x + \alpha y)$$

Geometrically, a function f is concave if the cord between any two points on the function lies everywhere on or below the function itself.

Consider a list of functions $f_i : X \to \mathbb{R}$ for $i = 1, .., n$ and a list of numbers $\alpha_1, .., \alpha_n$. The function $f = \sum_{i=1}^n \alpha_i f_i$ is called a linear combination of $f_1, .., f_n$. If each of the weights $\alpha_i \geqslant 0$, then f is a non-negative linear combination of $f_1, .., f_n$.

Theorem 14.3.2 *Suppose $f_1, .., f_n$ are concave functions and $(\alpha_1, .., \alpha_n) \geqslant 0$. Then, $f = \sum_{i=1}^n \alpha_i f_i$ is also a concave function. If at least one f_j is also strictly concave and $\alpha_j > 0$, then f is strictly concave.*

Even though a concave function need not be differentiable everywhere, the right and left hand derivatives always exist on the interior of the domain and $f^-(x) \geqslant f^+(x)$. As a result, f is both right and left continuous and therefore continuous. However, concave functions need not be continuous at the boundary.

For differentiable functions, the following theorem provides a simple necessary and sufficient conditions for concavity.

Theorem 14.3.3 *Suppose $f : X \to \mathbb{R}$ is differentiable.*

1. *f is concave if and only if for each $x, y \in X$ we have*

$$f(y) - f(x) \leqslant f^{'}(x)(y - x)$$

2. *f is strictly concave if and only if the inequality is strict for each $x \neq y$.*

Even if a function is not differentiable everywhere, it is concave if and only if for each $x \in int(X)$, there is an $a \in \mathbb{R}$ such that $f(y) - f(x) \leqslant a(y - x)$ for all $y \in X$. This is an example of a supporting hyperplane for one dimension.

Theorem (14.3.3) implies that the first derivative function of a concave function is non-increasing.

Theorem 14.3.4 *Suppose $f : X \to \mathbb{R}$ is differentiable.*

1. *f is concave if and only if $f^{'}$ is non-increasing.*
2. *f is strictly concave if and only if $f^{''}$ is strictly decreasing.*

Theorem 14.3.5 *Suppose $f : X \to \mathbb{R}$ is twice differentiable.*

1. *f is concave if and only if $f^{''} \leqslant 0$.*
2. *if $f^{''} < 0$, then f is strictly concave.*

Electronic copy available at: https://ssrn.com/abstract=3307566

Note, f strictly concave does not imply that $f^{''}(x) < 0$ for all x. An example of strictly concave function $f : \mathbb{R}_{++} \to \mathbb{R}$ is

1. $f(x) = \frac{x^\alpha}{\alpha}$ for $\alpha \neq 0, \alpha < 1$
2. $f(x) = \log x$
3. $f(x) = bx - ax^2$ where $a > 0$

The following lemma is an immediate consequence of the definition of concave and convex functions.

Lemma 14.3.1 *f is a (strictly) convex function if and only if $-f$ is a (strictly) concave function.*

14.3.3 Some approximations

1. Upper bound on the exponential function obtained by linearising e^x in 0
$$e^x \geqslant 1 + x$$
2. Lower bound on the logarithm function, derived by using the log operator on
$$\log(1 + x) \geqslant x$$
3. This inequality follows from the convexity of $f(z) = e^{\alpha z}$
$$e^{\alpha x} \leqslant 1 + (e^\alpha - 1)x \text{ for } x \in [0, 1]$$
4. Another inequality
$$-\log(1 - x) \leqslant x + x^2 \text{ for } x \in [0, \frac{1}{2}]$$

14.4 Introduction to wavelet theory

Fitting financial time series is a difficult task as they are not stationary and contain noise. It is possible to get rid of this noise by using a filter. However, when denoising a signal we need to preserve the relevant multi-scaled information, which can be done with wavelets.
We present a short introduction to wavelets and provide some examples on denoising techniques.

14.4.1 Some definitions

Wavelet theory overcomes some problems linked to Fourier theory (such as stationarity) by generalising the decomposition to a time-frequency domain. The word wavelet (ondelette) were used by Morlet and Grossmann in the early 1980s. Started with the work of Haar in the early 20th century, Zweig discovered of the continuous wavelet transform in 1975, which were extended by Goupillaud, Grossmann and Morlet. Stromberg introduced discrete wavelets (1983) and Daubechies presented the orthogonal wavelets with compact support (1988). Wavelets were first used in physics to characterise the Brownian motion and have been widely used in diverse tasks related to financial time series analysis.

Definition 14.4.1 *Wavelet*
A wavelet is a square integrable function $\psi_{s,\tau} : t \in \mathbb{R} \to \mathbb{R}$, where $(s, \tau) \in \mathbb{R}^{*+} \times \mathbb{R}$, such that $\psi_{s,\tau} \in L^2(\mathbb{R})$ and $\int_{-\infty}^{+\infty} \psi_{s,\tau}(t)dt = 0$.

***Remark* 14.4.1** *Wavelets derived from a mother wavelet Ψ in the following way: $\forall t \in \mathbb{R}, \psi_{s,\tau}(t) = \frac{1}{\sqrt{s}}\Psi(\frac{t-\tau}{s})$. In that sense, a mother wavelet generates a subgroup Λ of the affine transformation group of $\mathbb{R}^n$.*

There are many wavelet families, each of them gathering wavelets having common properties.

Electronic copy available at: https://ssrn.com/abstract=3307566

14.4.1.1 Continuous Wavelet

As in Fourier theory, both continuous and discrete wavelet transforms exist. We first define the continuous wavelet transform (CWT) of a signal f as follows:

$$\forall (s,\tau) \in \mathbb{R}^{*+} \times \mathbb{R} \qquad g(s,\tau) = \int_{-\infty}^{+\infty} f(t)\psi_{s,\tau}^{*}(t)dt$$

$*$ stands for the complex conjugate, τ is a translation factor and s is a dilatation factor. To recover f from g, the following equation is used:

$$\forall t \in \mathbb{R} \qquad f(t) = \frac{1}{C}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\frac{1}{|s|^2}g(s,\tau)\psi_{s,\tau}(t)dsd\tau$$

where $C = \int_{-\infty}^{+\infty}\frac{|\hat{\Psi}(x)|}{|x|}dx$, $\hat{\Psi}$ being the Fourier transform of the mother wavelet Ψ.

An example of such wavelet is the Mexican hat wavelet. The continuous wavelet transform defines a priori an infinite number of coefficients which cannot be used in practice. For that reason, we will only make use of discrete wavelet transform (DWT) introduced below.

14.4.1.2 Discrete Wavelet

Consider a discrete time set $\mathbb{T} = [\![0, T-1]\!]$. We define a discrete wavelet on this set as follows:

$$\forall t \in \mathbb{T} \qquad \psi_{m,n}(t) = s_0^{-m/2}\psi(s_0^{-m}t - n\tau_0)$$

where s_0 and τ_0 are constant. Then, the scaling and wavelet functions of DWT are defined as follows:

$$\phi(2^m t) = \sum_{i=1}^{n} h_{m+1}(i)\phi(2^{m+1}t - i)$$

$$\psi(2^m t) = \sum_{i=1}^{n} g_{m+1}(i)\phi(2^{m+1}t - i)$$

where $(h_{m+1}(i))_{i\in[\![1,n]\!]}$ and $(g_{m+1}(i))_{i\in[\![1,n]\!]}$ are respectively the coefficient sequences of ϕ and ψ. From these functions, a discrete signal x can be written as:

$$x(t) = \sum_{i=1}^{n}\lambda_{m-1}(i)\phi(2^{m-1}t - i) + \sum_{i=1}^{n}\gamma_{m-1}(i)\psi(2^{m-1}t - i)$$

If $\{\psi_{m,n} : m, n \in \mathbb{Z}\}$ defines an orthonormal basis of $L^2(\mathbb{R})$, the following formula is used in order to recover the signal x.

$$\forall t \in \mathbb{T} \qquad x(t) = \sum_{m\in\mathbb{Z}}\sum_{n\in\mathbb{Z}}\langle x, \psi_{m,n}\rangle \cdot \psi_{m,n}(t)$$

***Remark* 14.4.2** *Orthonormality is an interesting property but not necessarily required for wavelets' construction. Compact support is also an useful property wavelets have as they can calibrate data efficiently, though it is not required neither.*

***Remark* 14.4.3** *Using a set of wavelets having the orthonormal property makes it easier to recover a discrete signal from its coefficients $(\langle x, \psi_{m,n}\rangle)_{m,n}$. This is the case of Haar wavelets for example.*

Electronic copy available at: https://ssrn.com/abstract=3307566

Wavelets take care of temporal characteristics of the regular factors described earlier as they are defined in a two dimensional domain: frequency and time resolution combinaison. Wavelets are indeed small waves located at different times and therefore give significant information about both time and frequency domains. This new degree of freedom provides the ability to capture trends in a local fashion.

14.4.2 Decomposition schemes

We introduce three decomposition schemes standing for different approaches of the wavelet transform introduced earlier.

14.4.2.1 Discrete Wavelet Transform

DWT is the most basic wavelet decomposition, Figure (14.1) shows how it can be done. H, L and H', L' respectively stands for high-pass, low-pass filters for wavelet decomposition and reconstitution. In the decomposition phase, L and H remove respectively the higher frequencies and the lower frequencies to give an approximation and a detail of signal x. The result is then downsampled by 2. The right part of the figure denotes the reserve process and is known as the reconstruction phase.

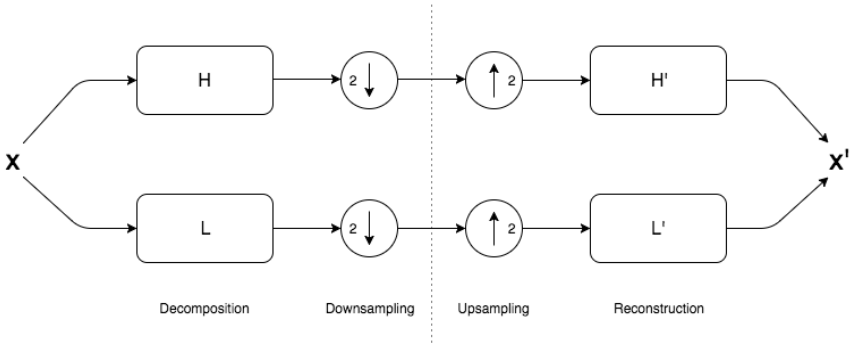


Figure 14.1: DWT Filter Scheme

More generally, we can repeat this process n times by a cascade algorithm which decomposes and downsamples the latter approximation. The following figure describes the process. A signal x can then be decomposed at level n in the following fashion:

$$x(t) = A_n(t) + \sum_{k=1}^{n} D_k(t) \tag{14.4.4}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

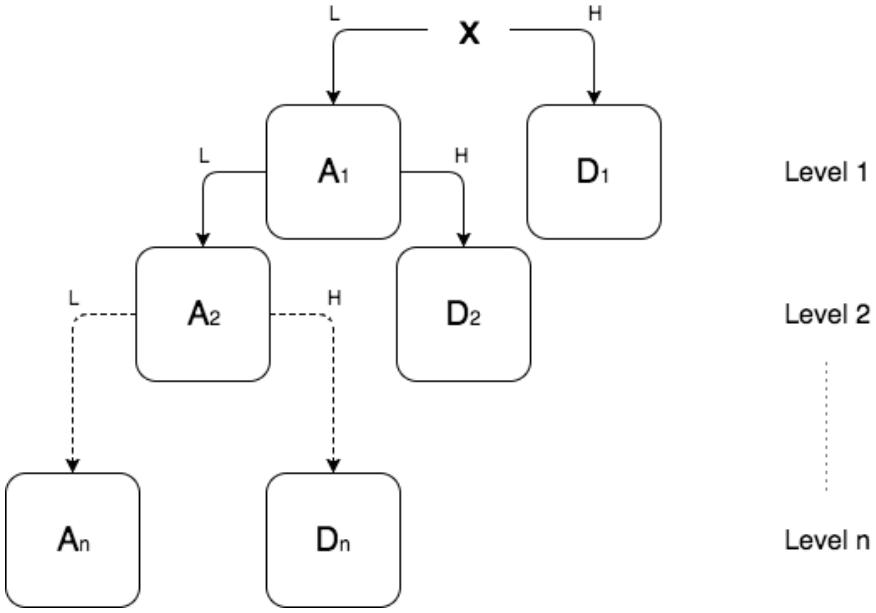


Figure 14.2: DWT Decomposition Tree

The main drawback of DWT is to be time-variant. For example, if we consider a periodic signal x with period τ, the DWT of x on $[\![t-\tau, t]\!]$ is in general not the DWT of x on $[\![t, t+\tau]\!]$ due to the downsampling. This property being desirable, researchers have developed a new type of wavelet transform called stationary wavelet transform (SWT) which guarantees time-invariance.

14.4.2.2 Stationary Wavelet Transform

In the DWT framework the downsampling meant choosing even or odd indexed elements at every decomposition step. However, in general, we could choose between even and odd at every decomposition which would lead to many possible combinaisons. The cornerstone of SWT is to average these denoised signals. We choose 0 to be coinciding with an even downsampling and 1 with an odd one.

Definition 14.4.2 *ϵ-decimated DWT*
Let N be a decomposition level of a signal x and $\epsilon = (\epsilon_1, ..., \epsilon_N)$ a sequence of integers such that $\forall n \in [\![1, N]\!]$, $\epsilon_n \in \{0, 1\}$. We say that the DWT $\chi = (A_N, D_N, .., D_1)$ is ϵ-decimated DWT if $\forall n \in [\![1, N]\!]$, ϵ_n corresponds to the downsampling choice of χ at time n.

It is possible to calculate all the ϵ-decimated DWTs for a given finite signal x by computing χ for every possible sequence ϵ. However, doing so is very computationally inefficient as 2^N DWT need to be done for a given decomposition level N. Another way to do this is to avoid downsampling and perform an upsampling on the original filters H and L.

Electronic copy available at: https://ssrn.com/abstract=3307566

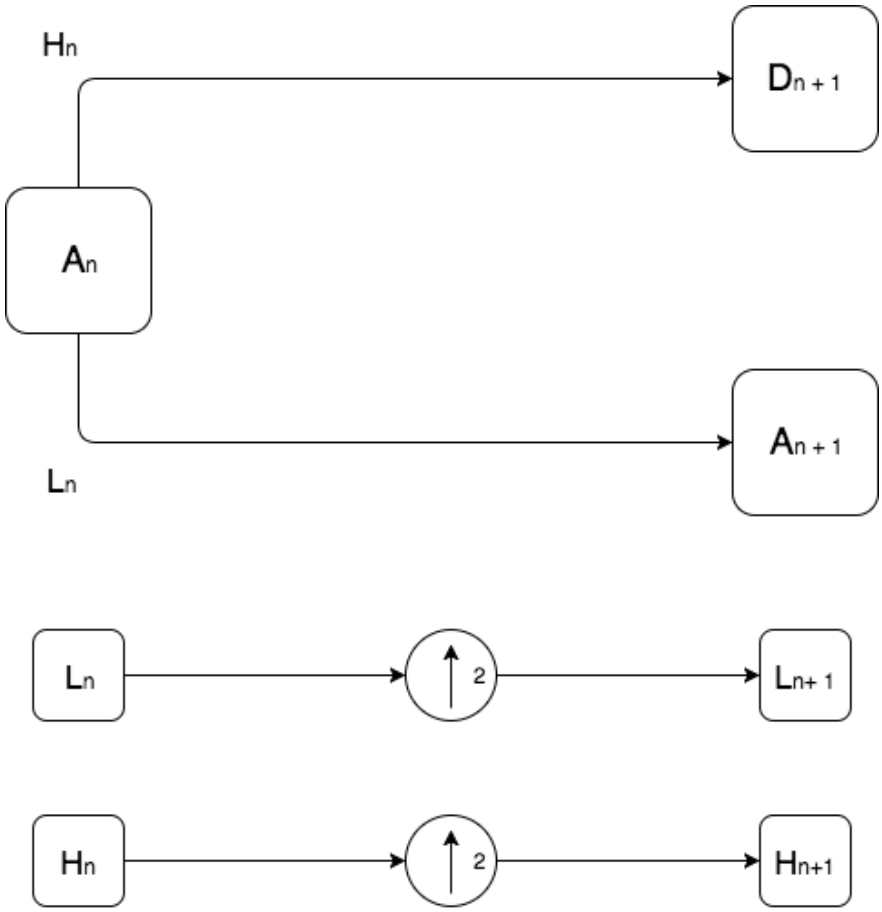


Figure 14.3: SWT calculation at level n

By following this process we end up with a time-invariant wavelet transform χ. In the next part, we describe a more general decomposition which is worth being investigated.

14.4.2.3 Wavelet Packet Transform

WPT can be seen as an extension of DWT. The latter decomposes only the approximation coefficients while WPT decomposes both the approximation and detail coefficients. This leads to more complex and somehow more flexible decomposition. Figure (14.4) describes its process:

Electronic copy available at: https://ssrn.com/abstract=3307566

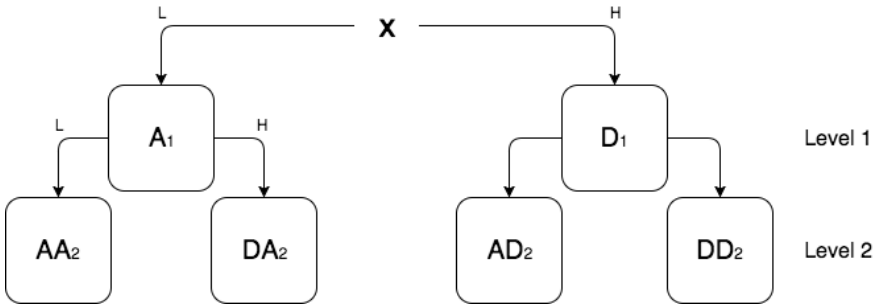


Figure 14.4: WPT Decomposition Tree

However, this process leads to 2^N coefficients for a given decomposition level N and thus can be more cumbersome in terms of storage and computation processing.

14.4.3 Denoising techniques

In this section we explain how DWT, SWT and WPT can be used to denoise a given signal. We say that a signal is noised when it suffers from unwanted or unknown modifications. Although we can have an intuitive understanding of noise, particularly in financial time series, there cannot be clear definition of it. Consequently, researchers often model it. A common model used in the research papers is the Gaussian white noise which has the characteristic to have the same power spectral density at every bandwidth frequency.

Definition 14.4.3 *Power spectral density*
Let x a signal and X its Fourier transform. We define the power spectral density (PSD) of the signal x as $\Gamma_x = |X|^2 * T$.

Definition 14.4.4 *Gaussian white noise*
A signal x is called a Gaussian white noise process if x is a stationary Gaussian process such that its mean $\mu_x = 0$ and its PSD is constant.

From theses definition we state our goal: from a noised signal x defined on a domain D and carrying a Gaussian white noise ϵ, we wish to find the true signal x^* such that:

$$\forall t \in D \quad x(t) = x^*(t) + \epsilon(t) \tag{14.4.5}$$

In the previous section, we have introduced the concept of wavelet decomposition leading to a set of approximation and detail coefficients. As the noise in a signal is generally contained in the detail ones, a common procedure consists in setting the smallest coefficients to zero. Given a threshold $\lambda \in \mathbb{R}^+$, we sum up the many possibilities below.

Electronic copy available at: https://ssrn.com/abstract=3307566

soft	hard
$x \to \begin{cases} x-\lambda & \text{if} \quad x > \lambda \\ 0 & \text{if} \quad \lvert x \rvert \leqslant \lambda \\ x+\lambda & \text{if} \quad x < -\lambda \end{cases}$	$x \to \begin{cases} 0 & \text{if} \quad \lvert x \rvert < \lambda \\ x & \text{otherwise} \end{cases}$
greater	**less**
$x \to \begin{cases} 0 & \text{if} \quad x < \lambda \\ x & \text{otherwise} \end{cases}$	$x \to \begin{cases} 0 & \text{if} \quad x > \lambda \\ x & \text{otherwise} \end{cases}$

Table 14.1: Threshold Function Definitions

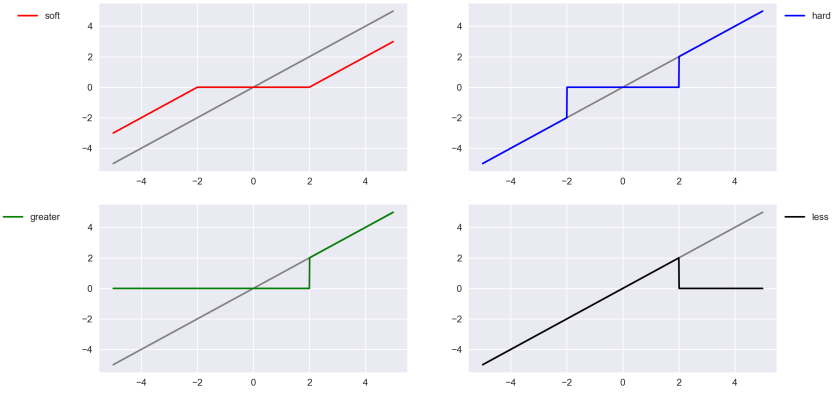


Figure 14.5: Thresholding functions ($\lambda = 2$)

It is a well-known fact that soft thresholding provides smoother results when comparing to other ones. However, hard thresholding provides better edge preservation.

Algorithm (38) defines our filtering process for Discrete Wavelet Transform and has been used in our experiments to denoise a signal.
Filtering processes for other wavelet decomposition methods are very similar, the general idea being thresholding detail coefficients.

14.5 Some random sampling

For details see text book by Rice [1995]. We assume that the population is of size N and that associated with each member of the population is a numerical value of interest denoted by $x_1, x_2, .., x_N$. The variable x_i may be a numerical variable such as age or weight, or it may take on the value 1 or 0 to denote the presence or absence of some characteristic. The latter is called the dichotomous case.

Electronic copy available at: https://ssrn.com/abstract=3307566

Algorithm 38 DWT filtering process

Require: get noised signal x
Require: set wavelet name wlt
Require: set decomposition level n
Require: set thresholding function f and level λ
1: $[A(n), D_n, D_{n-1}, .., D_1] \leftarrow$ decomposition(x, wlt, n)
2: **for** i = 1 **to** n **do**
3: $D_i \leftarrow f(D_i, \lambda)$
4: **end for**
5: $x^* \leftarrow$ reconstruction($[A, D]$, wlt)
6: **return** x^*

14.5.1 The sample moments

In general, the population variance of a finite population of size N is given by

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 \text{ with } \mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

where μ is the population mean. In the dichotomous case μ equals the proportion p of individuals in the population having the particular characteristic. The population total is

$$\tau = \sum_{i=1}^{N} x_i = N\mu$$

In many practical situations, the true variance of a population is not known a priori and must be computed somehow. When dealing with extremely large populations, it is not possible to count every object in the population and one must estimate the variance of a population from a sample. We take a sample with replacement of n values $X_1, ..., X_n$ from the population, where $n < N$ and such that X_i is a random variable. X_i is the value of the ith member of the sample, and x_i is that of the ith member of the population. The joint distribution of the X_i is determined by that of the x_i. We let $\xi_1, .., \xi_m$ be the elements of a vector corresponding to the possible values of x_i. Since each member of the population is equally likely to be in the sample, we get

$$P(X_i = \xi_j) = \frac{n_j}{N}$$

The sample mean is

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

Theorem 14.5.1 *With simple random sampling* $E[\overline{X}] = \mu$

We say that an estimate is unbiased if its expectation equals the quantity we wish to estimate.

$$\text{Mean squared error } = \text{ variance } + (biased)^2$$

Since $\overline{X}$ is unbiased, its mean square error is equal to its variance.

Lemma 14.5.1 *With simple random sampling*

$$Cov(X_i, X_j) = \begin{cases} \sigma^2 \text{ if } i = j \\ -\frac{\sigma^2}{(N-1)} \text{ if } i \neq j \end{cases}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

It shows that X_i and X_j are not independent of each other for $i \neq j$, but that the covariance is very small for large values of N.

Theorem 14.5.2 *With simple random sampling*

$$Var(\overline{X}) = \frac{\sigma^2}{n}\big(1 - \frac{n-1}{N-1}\big)$$

Note, the ratio $\frac{n}{N}$ is called the sampling fraction and

$$p_c = \big(1 - \frac{n-1}{N-1}\big)$$

is the finite population correction. If the sampling fraction is very small we get the approximation

$$\sigma_{\overline{X}} \approx \frac{\sigma}{\sqrt{n}}$$

We estimate the variance on the basis of this sample as

$$\hat{\sigma}^2 = \sigma_n^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^2 \text{ with } \overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

This is the biased sample variance. The unbiased sample variance is

$$\overline{\sigma}_n^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2 \text{ with } \overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

While the first one may be seen as the variance of the sample considered as a population (over n), the second one is the unbiased estimator of the population variance (over N) where $(n-1)$ is the Bessel's correction. Put another way, we get

$$E[\sigma_n^2] = \frac{n-1}{n}\frac{N}{N-1}\sigma^2 \text{ and } E[\overline{\sigma}_n^2] = \sigma^2$$

That is, the unbiased estimate of σ^2 may be obtained by multiplying σ_n^2 by the factor $\frac{n}{n-1}\frac{N-1}{N}$. If the population is large relative to n, the dominant bias is due to the term $\frac{n-1}{n}$. In general one set n between 50 and 180 days. Being a function of random variables, the sample variance is itself a random variable, and it is natural to study its distribution. In the case that y_i are independent observations from a normal distribution, Cochran's theorem shows that $\overline{\sigma}_n^2$ follows a scaled chi-squared distribution

$$(n-1)\frac{\overline{\sigma}_n^2}{\sigma^2} \approx \chi_{n-1}^2$$

If the conditions of the law of large numbers hold for the squared observations, $\overline{\sigma}_n^2$ is a consistent estimator of σ^2 and the variance of the estimator tends asymptotically to zero. The obtained standard deviation $\tilde{\sigma}_N$ is an estimator of σ called the historical volatility.

***Corollary* 4** *An unbiased estimate of* $Var(\overline{X})$ *is*

$$s_{\overline{X}}^2 = \frac{\hat{\sigma}^2}{n}\frac{n}{n-1}\frac{N-1}{N}\frac{N-n}{N-1} = \frac{s^2}{n}(1 - \frac{n}{N})$$

where

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2$$

Electronic copy available at: https://ssrn.com/abstract=3307566

One can compute the sample mean and sample variance recursively as follow. Starting from the identities

$$(n+1)\overline{X}_{n+1} = n\overline{X}_n + X_{n+1}$$

and

$$(n+1)\big(\sigma_{n+1}^2 + (\overline{X}_{n+1})^2\big) = n\big(\sigma_n^2 + (\overline{X}_n)^2\big) + X_{n+1}^2$$

we get the recursive sample mean

$$\overline{X}_{n+1} = \overline{X}_n + \frac{X_{n+1} - \overline{X}_n}{n+1} \tag{14.5.6}$$

and the recursive sample variance

$$\sigma_{n+1}^2 = \sigma_n^2 + (\overline{X}_n)^2 - (\overline{X}_{n+1})^2 + \frac{X_{n+1}^2 - \sigma_n^2 - (\overline{X}_n)^2}{n+1} \tag{14.5.7}$$

such that σ_{n+1}^2 only depends on σ_n^2, $\overline{X}_n$, $\overline{X}_{n+1}$ and X_{n+1}. If we normalise the data with a constant K such that the ith value is $\frac{X_i}{K}$, we let $\tilde{X} = \frac{1}{n}\sum_{i=1}^n \frac{X_i}{K}$ be the sample mean and $\tilde{\sigma}_n^2 = \frac{1}{n}\sum_{i=1}^n (\frac{X_i}{K} - \tilde{X})^2$ be the sample variance. Then, the recursive sample mean becomes

$$\tilde{X}_{n+1} = \tilde{X}_n + \frac{1}{n+1}\big(\frac{X_{n+1}}{K} - \tilde{X}_n\big) \tag{14.5.8}$$

and the recursive sample variance becomes

$$\tilde{\sigma}_{n+1}^2 = \tilde{\sigma}_n^2 + (\tilde{X}_n)^2 - (\tilde{X}_{n+1})^2 + \frac{1}{n+1}\Big(\frac{X_{n+1}^2}{K^2} - \tilde{\sigma}_n^2 - (\tilde{X}_n)^2\Big) \tag{14.5.9}$$

14.5.2 Estimation of a ratio

We consider the estimation of a ratio, and assume that for each member of a population, two values, x and y, may be measured. The ratio of interest is

$$r = \frac{\sum_{i=1}^N y_i}{\sum_{i=1}^N x_i} = \frac{\mu_y}{\mu_x}$$

Assuming that a sample is drawn consisting of the pairs (X_i, Y_i), the natural estimate of r is $R = \frac{\overline{Y}}{\overline{X}}$. Since R is a nonlinear function of the random variables $\overline{X}$ and $\overline{Y}$, there is no closed form for $E[R]$ and $Var(R)$ and we must approximate them by using $Var(\overline{X})$, $Var(\overline{Y})$, and $Cov(\overline{X}, \overline{Y})$. We define the the population covariance of x and y as

$$\sigma_{xy} = \frac{1}{N}\sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y)$$

One can show that

$$Cov(\overline{X}, \overline{Y}) = \frac{\sigma_{xy}}{n} p_c$$

Theorem 14.5.3 *With simple random sampling, the approximate variance of $R = \frac{\overline{Y}}{\overline{X}}$ is*

$$Var(R) \approx \frac{1}{\mu_x^2}(r^2\sigma_{\overline{X}}^2 + \sigma_{\overline{Y}}^2 - 2r\sigma_{\overline{XY}}) = \frac{1}{n}p_c\frac{1}{\mu_x^2}(r^2\sigma_x^2 + \sigma_y^2 - 2r\sigma_{xy})$$

Electronic copy available at: https://ssrn.com/abstract=3307566

The population correlation coefficient given by

$$\rho = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

is a measure of the strength of the linear relationship between the x and the y values in the population. We can express the variance in the above theorem in terms of ρ as

$$Var(R) \approx \frac{1}{n} p_c \frac{1}{\mu_x^2}(r^2\sigma_x^2 + \sigma_y^2 - 2r\rho\sigma_x\sigma_y)$$

so that strong correlation of the same sign as r decreases the variance. Further, for small μ_x we get large variance, since small values of $\overline{X}$ in the ratio $R = \frac{\overline{Y}}{\overline{X}}$ cause R to fluctuate wildly.

Theorem 14.5.4 *With simple random sampling, the expectation of R is given approximately by*

$$E[R] \approx r + \frac{1}{n} p_c \frac{1}{\mu_x^2}(r^2\sigma_x^2 - \rho\sigma_x\sigma_y)$$

so that strong correlation of the same sign as r decreases the bias, and the bias is large if μ_x is small. In addition, the bias is of the order $\frac{1}{n}$, so its contribution to the mean squared error is of the order $\frac{1}{n^2}$ while the contribution of the variance is of order $\frac{1}{n}$. Hence, for large sample, the bias is negligible compared to the standard error of the estimate. For large samples, truncating the Taylor Series after the linear term provides a good approximation, since the deviations $\overline{X} - \mu_x$ and $\overline{Y} - \mu_y$ are likely to be small. To this order of approximation, R is expressed as a linear combination of $\overline{X}$ and $\overline{Y}$, and an argument based on the central limit theorem can be used to show that R is approximately normally distributed, and confidence intervals can be formed for r by using the normal distribution. In order to estimate the standard error of R, we substitute R for r in the formula of the above theorem where the x and y population variances are estimated by s_x^2 and s_y^2. The population covariance is estimated by

$$s_{xy} = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y}) = \frac{1}{n-1}\Big(\sum_{i=1}^{n} X_i Y_i - n\overline{XY}\Big)$$

and the population correlation is estimated by

$$\hat{\rho} = \frac{s_{xy}}{s_x s_y}$$

The estimated variance of R is thus

$$s_R^2 = \frac{1}{n} p_c \frac{1}{\overline{X}^2}(R^2 s_x^2 + s_y^2 - 2R s_{xy})$$

and the approximate $100(1-\alpha)\%$ confidence interval for r is $R \pm z(\frac{\alpha}{2})s_R$.

14.5.3 Stratified random sampling

The population is partitioned into subpopulations, or strata, which are then independently sampled. We are interested in obtaining information about each of a number of natural subpopulations in addition to information about the population as a whole. It guarantees a prescribed number of observations from each subpopulation, whereas the use of a simple random sample can result in underrepresentation of some subpopulations. Further, the stratified sample mean can be considerably more precise than the mean of a simple random sample, especially if there is considerable variation between strata.

We will denote by N_l, where $l = 1, .., L$ the population sizes in the L strata such that $N_1 + N_2 + .. + N_L = N$ the total population size. The population mean and variance of the lth stratum are denoted by μ_l and σ_l^2 (unknown). The overall population mean can be expressed in terms of the μ_l as follows

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\mu = \frac{1}{N}\sum_{l=1}^{L}\sum_{i=1}^{N_l} x_{il} = \frac{1}{N}\sum_{l=1}^{L} N_l\mu_l = \sum_{l=1}^{L} W_l\mu_l$$

where x_{il} denotes the ith population value in the lth stratum and $W_l = \frac{N_l}{N}$ is the fraction of the population contained in the lth stratum.

Within each stratum a simple random sample of size n_l is taken. The sample mean in stratum l is denoted by

$$\overline{X}_l = \frac{1}{n_l}\sum_{i=1}^{n_l} X_{il}$$

where X_{il} denotes the ith observation in the lth stratum. By analogy with the previous calculation we get

$$\overline{X}_s = \sum_{l=1}^{L} \frac{N_l\overline{X}_l}{N} = \sum_{l=1}^{L} W_l\overline{X}_l$$

Theorem 14.5.5 *The stratisfied estimate, $\overline{X}_s$ of the population mean is unbiased. That is, $E[\overline{X}_s] = \mu$.*

Since we assume that the samples from different strata are independent of one another and that within each stratum a simple random sample is taken, the variance of $\overline{X}_s$ can easily be calculate.

Theorem 14.5.6 *The variance of the stratisfied sample mean is given by*

$$Var(\overline{X}_s) = \sum_{l=1}^{L} W_l^2 \frac{1}{n_l}(1 - \frac{n_l - 1}{N_l - 1})\sigma_l^2 \tag{14.5.10}$$

Note, $\frac{n_l-1}{N_l-1}$ represents the finite population corrections. If the sampling fractions within all strata are small ($\frac{n_l-1}{N_l-1} << 1$), we get the approximation

$$Var(\overline{X}_s) \approx \sum_{l=1}^{L} \frac{W_l^2}{n_l}\sigma_l^2 \tag{14.5.11}$$

The estimate of σ_l^2 is given by

$$S_l^2 = \frac{1}{n_l - 1}\sum_{i=1}^{n_l}(X_{il} - \overline{X}_l)^2$$

and $Var(\overline{X}_s)$ is estimated by

$$S^2_{\overline{X}_s} = \sum_{l=1}^{L} W_l^2 \frac{1}{n_l}(1 - \frac{n_l - 1}{N_l - 1})S_l^2$$

The question that naturally arises is how to choose $n_1, .., n_L$ to minimise $Var(\overline{X}_s)$ subject to the constraint $n_1 + .. + n_L = n$. Ignoring the finite population correction within each stratum we get the Neyman allocation.

Theorem 14.5.7 *The sample sizes $n_1, .., n_L$ that minimise $Var(\overline{X}_s)$ subject to the constraint $n_1 + .. + n_L = n$ are given by*

$$n_l = n\frac{W_l\sigma_l}{\sum_{k=1}^{L} W_k\sigma_k}$$

where $l = 1, .., L$.

Electronic copy available at: https://ssrn.com/abstract=3307566

This theorem shows that those strata for which $W_l\sigma_l$ is large should be sampled heavily. If W_l is large, the stratum contains a large fraction of the population. If σ_l is large, the population values in the stratum are quite variable, and a relatively large sample size must be used. Substituting the optimal values of n_l into the Equation (14.5.10) we get the following corollary.

Corollary 5 *Denoting by $\overline{X}_{so}$ the stratified estimate using the optimal Neyman allocation and neglecting the finite population correction, we get*

$$Var(\overline{X}_{so}) = \frac{1}{n}\Big(\sum_{l=1}^{L} W_l\sigma_l\Big)^2$$

The optimal allocations depend on the individual variances of the strata, which generally will not be known. A simple alternative method of allocation is to use the same sampling fraction in each stratum

$$\frac{n_1}{N_1} = \frac{n_2}{N_2} = ... = \frac{n_L}{N_L}$$

which holds if

$$n_l = n\frac{N_L}{N} = nW_l \tag{14.5.12}$$

$l = 1, .., L$. This method is called the Proportional allocation. The estimate of the population mean based on proportional allocation is

$$\overline{X}_{sp} = \sum_{l=1}^{L} W_l\overline{X}_l = \frac{1}{n}\sum_{l=1}^{L}\sum_{i=1}^{n_l} X_{il}$$

since $\frac{W_l}{n_l} = \frac{1}{n}$. This estimate is simply the unweighted mean of the sample values.

Theorem 14.5.8 *With stratified sampling based on proportional allocation, ignoring the finite population correction, we get*

$$Var(\overline{X}_{sp}) = \frac{1}{n}\sum_{l=1}^{L} W_l\sigma_l^2$$

We can compare $Var(\overline{X}_{so})$ and $Var(\overline{X}_{sp})$ to define when optimal allocation is substantially better than proportional allocation.

Theorem 14.5.9 *With stratified random sampling, the difference between the variance of the estimate of the population mean based on proportional allocation and the variance of that estimate based on optimal allocation is, ignoring the finite population correction,*

$$Var(\overline{X}_{sp}) - Var(\overline{X}_{so}) = \frac{1}{n}\sum_{l=1}^{L} W_l(\sigma_l - \overline{\sigma})^2$$

where

$$\overline{\sigma} = \sum_{l=1}^{L} W_l\sigma_l$$

Electronic copy available at: https://ssrn.com/abstract=3307566

As a result, if the variances of the strata are all the same, proportional allocation yields the same results as optimal allocation. The more variable these variances are, the better it is to use optimal allocation.

We can also compare the variance under simple random sampling with the variance under proportional allocation. Neglecting the finite population correction, the variance under simple random sampling is $Var(\overline{X}) = \frac{\sigma^2}{n}$. We first need a relationship between the overall population variance σ^2 and the strata variances σ_l^2. The overall population variance may be expressed as

$$\sigma^2 = \frac{1}{N}\sum_{l=1}^{L}\sum_{i=1}^{N_L}(x_{il} - \mu)^2$$

Further, using the relation

$$(x_{il} - \mu)^2 = (x_{il} - \mu_l)^2 + 2(x_{il} - \mu_l)(\mu_l - \mu) + (\mu_l - \mu)^2$$

and realising that when both sides are summed over l, the middle term on the right-hand side becomes zero, we have

$$\sum_{i=1}^{N_L}(x_{il} - \mu)^2 = \sum_{i=1}^{N_L}(x_{il} - \mu_l)^2 + N_L(\mu_l - \mu)^2 = N_L\sigma_l^2 + N_L(\mu_l - \mu)^2$$

Dividing both sides by N and summing over l, we have

$$\sigma^2 = \sum_{l=1}^{L} W_l\sigma_l^2 + \sum_{l=1}^{L} W_l(\mu_l - \mu)^2$$

Substituting this expression for σ^2 into $Var(\overline{X})$ and using the formula for $Var(\overline{X}_{sp})$ completes the proof of the following theorem.

Theorem 14.5.10 *The difference between the variance of the mean of a simple random sample and the variance of the mean of a stratified random sample based on proportional allocation is, neglecting the finite population correction,*

$$Var(\overline{X}) - Var(\overline{X}_{sp}) = \frac{1}{n}\sum_{l=1}^{L} W_l(\mu_l - \mu)^2$$

Thus, stratified random sampling with proportional allocation always gives a smaller variance than does simple random sampling, providing that the finite population correction is ignored. Typically, stratified random sampling can result in substantial increases in precision for populations containing values that vary greatly in size.

In order to construct the optimal number of strata, the population values themselves (which are unknown) would have to be used. Stratification must therefore be done on the basis of some related variable that is known or on the results of earlier samples.

According to the Neyman-Pearson Paradigm, a decision as to weather or not to reject H_0 in favour of H_A is made on the basis of $T(X)$, where X denotes the sample values and $T(X)$ is a statistic.

The statistical properties of the methods are relevant if it is reasonable to model the data stochastically. There exists methods that are sample analogues of the cumulative distribution function of a random variable. It is useful in displaying the distribution of sample values.
Given $x_1, .., x_n$ a batch of numbers, the empirical cumulative distribution function (ecdf) is defined as

$$F_n(x) = \frac{1}{n}(nox_i \leqslant x)$$

Electronic copy available at: https://ssrn.com/abstract=3307566

where $F_n(x)$ gives the proportion of the data less than or equal to x. It is a step function with a jump of height $\frac{1}{n}$ at each point x_i. The ecdf is to a sample what the cumulative distribution is to a random variable. We now consider some of the elementary statistical properties of the ecdf when $X_1, .., X_n$ is a random sample from a continuous distribution function F. We choose to express F_n as

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{(-\infty,x]}(X_i)$$

The random variables $I_{(-\infty,x]}(X_i)$ are independent Bernoulli random variables

$$I_{(-\infty,x]}(X_i) = \begin{cases} 1 \text{ with probability } F(x) \\ 0 \text{ with probability } 1 - F(x) \end{cases}$$

Thus, $nF_n(x)$ is a binomial random variable with the first two moments being

$$\begin{aligned} E[F_n(x)] &= F(x) \\ Var(F_n(x)) &= \frac{1}{n}F(x)[1 - F(x)] \end{aligned}$$

14.5.4 Geometric mean

The use of a geometric mean normalises the ranges being averaged, so that no range dominates the weighting, and a given percentage change in any of the properties has the same effect on the geometric mean. The geometric mean applies only to positive numbers. It is also often used for a set of numbers whose values are meant to be multiplied together or are exponential in nature, such as data on the growth of the human population or interest rates of a financial investment. The geometric mean of a data set $\{a_1, a_2, .., a_n\}$ is given by

$$(\prod_{i=1}^{n} a_i)^{\frac{1}{n}}$$

The geometric mean of a data set is less than the data set's arithmetic mean unless all members of the data set are equal, in which case the geometric and arithmetic means are equal. By using logarithmic identities to transform the formula, the multiplications can be expressed as a sum and the power as a multiplication

$$(\prod_{i=1}^{n} a_i)^{\frac{1}{n}} = e^{\frac{1}{n}\sum_{i=1}^{n} \ln a_i}$$

This is sometimes called the log-average. It is simply computing the arithmetic mean of the logarithm-transformed values of a_i (i.e., the arithmetic mean on the log scale) and then using the exponentiation to return the computation to the original scale.

14.5.5 Stochastic approximation

We can approximate the Bellman optimality equation by replacing the expectation with its empirical average (see Appendix (14.5.1)). Rather than directly summing the samples, Robbins et al. [1951] proposed to add data points one by one, and iteratively update the running estimate of the mean $\widehat{x}_n$, where n is the number of iteration or the number of data points in the dataset. It is an on-line algorithm. The update equation is

$$\widehat{x}_{n+1} = (1 - \alpha_n)\widehat{x}_n + \alpha_n x_n$$

where x_n is the n-th data point and $\alpha_n <$ is the learning rate that should satisfy the conditions

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\lim_{N\to\infty}\sum_{n=1}^{N}\alpha_n = \infty \text{ and } \lim_{N\to\infty}\sum_{n=1}^{N}\left(\alpha_n\right)^2 < \infty$$

The authors showed that under these constraints, the iterative method of computing the mean converges to a true mean with probability one. In general, the optimal choice of the learning rate α_n is problem dependent.

14.5.6 Accounting for time and earning profits

In traditional time series forecasting, the criteria for assessing model performance are error functions based on the goodness of fit of target and predicted values. Since neural networks are similar to conventional regression estimators, except for their nonlinearity, error functions are also used to judge the goodness of fit of the model. However, normalised mean square error (NMSE) and other error functions are not the most appropriate measures in finance.

14.5.6.1 Time factor

For instance, weighting all data equally (ordinary leasy squares) are less accurate than discounted least squares, which weight the most recent data more heavily (see Makridakis et al. [1982]). Incorporating time factor to neural networks, Refenes et al. [1997] proposed the discounted least squares (DLS) neural network model. Assuming a feed-forward network with n input and a single output unit ($m = 1$) with k hidden layers, we let the ordinary least-squares criterion of the network be given by

$$E_{LS} = \frac{1}{2P}\sum_{p=1}^{P}(O_p - d_p)^2$$

where O_p and d_p denote the pth output and target value, respectively.

***Remark* 14.5.1** *In general, when considering a training set consisting of sequential data, or time-varying input, we represent them as $\{(\overline{x}_1, \overline{d}_1), ..., (\overline{x}_T, \overline{d}_T)\}$ with T ordered pairs. To be consistent with the articles described we keep the P ordered pairs, and we let $p = P$ be the most recent observation.*

Refenes et al. [1997] proposed a simple modification to the error backpropagation procedure taking into account gradually changing input-output pairs. The procedure is based on the principle of discounted least-squares whereby learning is biased towards more recent observations with long term effects experiencing exponential decay through time. The cumulative error calculated by the DLS procedure is given by

$$E_{DLS} = \frac{1}{2P}\sum_{p=1}^{P}\phi(p)(O_p - d_p)^2$$

where $\phi(p)$ is an adjustment of the contribution of observation p to the overall error. They chose the simple sigmoidal decay function

$$\phi_{DLS}(p) = \frac{1}{1 + e^{a-bp}}$$

where $b = \frac{2a}{P}$ and a is the discount rate. The learning rule is derived in the usual way by repeatedly changing the weights by an amount proportional to

$$\frac{\partial E_{DLS}}{\partial W} = \phi_{DLS}(p)\frac{\partial E_{LS}}{\partial W}$$

since the discount factor $\phi(p)$ is a function of the recency of the observation which is independent of the actual order of pattern presentation within the learning process. Hence, the algorithm is not affected by randomising the order in

Electronic copy available at: https://ssrn.com/abstract=3307566

which patterns are presented and can be applied to both batch and stochastic update rules. Using a sine wave with changing amplitude and a non-trivial application in exposure analysis of stock returns to multiple factors, Refenes et al. [1997] compared the performance of the two cost functions on a backpropagation network with a single hidden layer of 12 units and batch update. They showed that DLS was a more efficient procedure for weakly non-stationary data series.

14.5.6.2 Direction measures

Further, we are usualy more interested in earning profits than in the quality of the forecast itself. When classifying returns, it is possible to use a direction measure to test the number of times a prediction neural network predicts the direction of the predicted return movement correctly. Merton [1981] proposed a modified direction given by

$$\text{Modified Direction} = \frac{\sharp \text{ of correct up predictions}}{\sharp \text{ of times index up}} + \frac{\sharp \text{ of correct down predictions}}{\sharp \text{ of times index down}} - 1$$

In general, direction is defined as the number of times the prediction followed the up and down movement of the underlying stock/index. Harvey et al. [2002] and Castiglione et al. [2001] proposed the following direction measure

$$\xi = \frac{1}{|T|} \sum_{t \in T} \mathcal{H}(R_t \hat{R}_t) + 1 - (|R_t| + |\hat{R}_t|)$$

where $R_t = \frac{P_t}{P_{t-1}}$ is the percentage return on the underlying at time step $t \in T$, $\hat{R}_t$ is the forecast percentage return, T is the number of days in the validation period, and $\mathcal{H}(\cdot)$ is the Heaviside function. Alternatively, we can write the direction measure as

$$\xi = \frac{1}{|T|} \sum_{t \in T} \mathcal{H}(\Delta P_t \Delta \hat{P}_t) + 1 - (|\Delta P_t| + |\Delta \hat{P}_{i+1}|)$$

where $\Delta P_t = P_t - P_{t-1}$ is the price change at step $t \in T$ and P_{t-1} is assumed to be known. These two direction measures provide a summary of how well the predicted time series and the actual ones move together at any point in time. However, we want to implement measurement errors taking into account the simultaneous behaviour of trend and magnitude within the trend. Hence, to reflect this point when evaluating the performance of a forecasting model, Yao et al. [1996] used the correctness of trend and a paper profit. They proposed a profit based adjusted weight factor for backpropagation network training by adding a factor containing profit, direction, and time information to the error function.

14.5.6.3 Time dependent direction profit

In order to take profit gain into account, Caldwell [1995] proposed a weighted directional symmetry (WDS) function which penalise more heavily the incorrectly predicted directions than the correct ones. The cumulative error function is given by

$$E_{WDS} = \frac{100}{P} \sum_{p=1}^{P} \phi(p) |O_p - d_p|$$

with

$$\phi(p) = \begin{cases} g \text{ if } \Delta d_p \times \Delta O_p \leqslant 0 \\ h \text{ otherwise} \end{cases}$$

where $\Delta d_p = d_p - d_{p-1}$ and $\Delta O_p = O_p - O_{p-1}$, and g and h are constants or some function of d_p. The values $g = 1.5$ and $h = 0.5$ were suggested. Yao et al. [1996] argued that the profit driven procedure was not only a function of the direction, but also of the amount of change. That is, the penalty on WDS weights should be increased in case

Electronic copy available at: https://ssrn.com/abstract=3307566

of a wrongly forecasted direction for a big change of values, and the penalty should be further reduced if the direction is correctly forecasted for a big change in value. They proposed a modified directional profit (DP) adjustment factor defined as

$$\phi_{DP}(p) = \begin{cases} a_1 \text{ if } \Delta d_p \times \Delta O_p > 0 \text{ and } |\Delta d_p| \leqslant \sigma \\ a_2 \text{ if } \Delta d_p \times \Delta O_p > 0 \text{ and } |\Delta d_p| > \sigma \\ a_3 \text{ if } \Delta d_p \times \Delta O_p < 0 \text{ and } |\Delta d_p| \leqslant \sigma \\ a_4 \text{ if } \Delta d_p \times \Delta O_p < 0 \text{ and } |\Delta d_p| > \sigma \end{cases}$$

where σ is a threshold for the changes in sample values generally taken as the standard deviation of the training set

$$\sigma^2 = \frac{1}{P}\sum_{p=1}^{P}(d_p - \mu_d)^2$$

where μ_d is the mean of the target series. The values $a_1 = 0.5$, $a_2 = 0.8$, $a_3 = 1.2$, and $a_4 = 1.5$ were suggested. While the DLS model focus on a time factor $\phi_{DLS}(p)$ and the DP model focus on a profit factor $\phi_{DP}(p)$, Yao et al. [1998] [2000] combined the two approaches obtaining a time dependent directional profit (TDP) model given by

$$\phi_{TDP}(p) = \phi_{DLS}(p) \times \phi_{DP}(p)$$

The three models, DLS, DP, and TDP were tested and compared to the benchmarked OLS model on four major Asian stock indices and the US Dow Jones Industrials Index (DJ). They considered a one hidden layer network with learning rate $\eta = 0.25$ and momentum rate $\gamma = 0.9$. The structure of the neural network was $30 - 10 - 1$, that is, 30 input, 10 nodes in the hidden layer and one output. Thus, 30 consecutive days of data are fed to the network to forecast the index of the following day. Considering a paper profit measure for the model performance, they showed that the preference of DLS over OLS was of 60%, and that the preference of DP over OLS was of 80%. In this combined model, they obtained up to 27.6% excess annual return above the OLS model with at least 2.8% excess annual return in the worst case. Lu Dang Khoa et al. [2006] argued that big changes in prices between two consecutive periods are rare to occur, leading to change between two consecutive prices smaller than the standard deviation of the price. They replaced the inequality $|\Delta d_p| > \sigma$ with $\frac{\Delta d_p}{d_{p-1}} > a_5$ and obtained the following adjustment factor

$$\phi_{DPN}(p) = \begin{cases} a_1 \text{ if } \Delta d_p \times \Delta O_p > 0 \text{ and } \frac{\Delta d_p}{d_{p-1}} < a_5 \\ a_2 \text{ if } \Delta d_p \times \Delta O_p > 0 \text{ and } \frac{\Delta d_p}{d_{p-1}} > a_5 \\ a_3 \text{ if } \Delta d_p \times \Delta O_p < 0 \text{ and } \frac{\Delta d_p}{d_{p-1}} < a_5 \\ a_4 \text{ if } \Delta d_p \times \Delta O_p < 0 \text{ and } \frac{\Delta d_p}{d_{p-1}} > a_5 \end{cases}$$

with $a_5 = 0.05$. They used that model to predict the $S\&P$ 500 stock index one month in the future without much improvements compared to the traditional FFN algorithm. They considered a mixed of input based on fundamental factors such as the prices of oil, the interest rates, inflation, and technical indicators such as volatility, relative strength index, directional index etc.

14.6 Some notion of probabilities

See textbook by Grimmett et al. [1992].

14.6.1 Events as sets and their probabilities

14.6.1.1 Sets of events

The occurrence or non-occurrence of an event A depends upon the chain of circumstances involved, called an experiment or trial. The result of an experiment is called its outcome. When it is non-predictable, we can only list the

Electronic copy available at: https://ssrn.com/abstract=3307566

collection of possible outcomes.

Definition 14.6.1 *The set of all possible outcomes of an experiment is called the sample space and is denoted* Ω.

Definition 14.6.2 *An event is a property which can be observed either to hold or not to hold after the experiment is done. In mathematical terms, an event is a subset of* Ω.

The complement of a subset A of Ω is denoted by A^c. When considering the events A and B, we will also be concerned with the events $A \cup B$, $A \cap B$ and A^c representing A or B, A and B and not A, respectively. Events A and B are called disjoint if their intersection is the empty set $\varnothing$ (called the impossible set). The set Ω is called the certain event. While events are subsets of Ω, all subsets of Ω are not events.

14.6.1.2 Probability measures

Given that we can repeat an experiment a large number N of times, keeping the initial conditions as equal as possible, we let A be some event which may or may not occur on each repetition. We are interested in the likelihoods of the occurrence of such an event when N increases and denote $P(A)$ the probability that A occurs on any particular trial. If A and B are two disjoint events, then

$$P(A \cup B) = P(A) + P(B)\,, P(\varnothing) = 0\,, P(\Omega) = 1$$

Thus, P should be countably additive, and one can define the desirable properties of a probability function P applied to the set of events as:

Definition 14.6.3 *Probability measure*
A probability measure P *on* $(\Omega, \mathcal{F})$ *is a function* $P : \mathcal{F} \to [0, 1]$ *satisfying*

1. $P(\varnothing) = 0$ *and* $P(\Omega) = 1$
2. *if* $A_1, A_2, ...,$ *is a collection of disjoint members of* $\mathcal{F}$*, so that* $A_i \cap A_j = \varnothing$ *for all pairs* i, j *satisfying* $i \neq j$*, then*

$$P(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$$

More generally, a measure μ assigns positive numbers to sets $A : \mu(A) \in \mathbb{R}$. For instance, if A is a subset of Euclidean space, then $\mu(A)$ is a length, area, or volume. If A is an event, then $\mu(A)$ is a probability of the event. If $\mathcal{X}$ is a space, then $\mu(\mathcal{X})$ measures the whole space. Measures are non-negative (signed measures need not be), zero on the empty set, and countably additive on disjoint sets. In general, it is not possible to assign a measure to every subset of an arbitrary space $\mathcal{X}$ in a way consistent with these axioms. Hence, to fully specify a measure space we need one more piece of data, the set of subsets of $\mathcal{X}$ which are measurable. These form a σ-algebra: they contain $\mathcal{X}$ and are closed under taking complements and countable unions (and hence, by De Morgan's laws, also closed under countable intersections). If we replace the word "countable" with "finite" we obtain the weaker notion of Jordan content, and if we replace it with "arbitrary" we get limp hogwash.

Electronic copy available at: https://ssrn.com/abstract=3307566

14.6.1.3 Conditional events

We now introduce the notion of conditional events:

Definition 14.6.4 *If $P(B > 0)$ then the conditional probability that A occurs given that B occurs is defined to be*

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

A family $B_1, B_2, ..., B_n$ of events is called a partition of Ω if

$$B_i \cap B_j = \varnothing \text{ when } i \neq j \text{ and } \bigcup_{i=1}^{n} B_i = \Omega$$

Each elementary $\omega \in \Omega$ belongs to exactly one set in a partition of Ω.

Lemma 14.6.1 *For any events A and B*

$$P(A) = P(A|B)P(B) + P(A|B^c)P(B^c)$$

More generally, let $B_1, B_2, ..., B_n$ be a partition of Ω. Then

$$P(A) = \sum_{i=1}^{n} P(A|B_i)P(B_i)$$

14.6.2 Defining random variables

A random variable can take on one of a set of different values, each with an associated probability. Its value at a particular time is subject to random variation.

Definition 14.6.5 *A random variable is a function $X : \Omega \to \mathbb{R}$ with the property that $\{\omega \in \Omega : X(\omega) \leqslant x\} \in \mathcal{F}$ for each $x \in \mathbb{R}$.*

Definition 14.6.6 *The distribution function of a random variable X is the function $F : \mathbb{R} \to [0, 1]$ given by*

$$F(x) = P(X \leqslant x)$$

14.6.2.1 Discrete random variables

Discrete random variables take on one of a discrete range of values.

Definition 14.6.7 *The random variable X is called discrete if it takes values in some countable subset $\{x_1, x_2, ..\}$, only, of $\mathbb{R}$.*

Its distribution function $F(x) = P(X \leqslant x)$ is a jump function.

Definition 14.6.8 *The probability mass function of a discrete random variable X is the function $f : \mathbb{R} \to [0, 1]$ given by $f(x) = P(X = x)$.*

The distribution and mass functions are related by

$$F(x) = \sum_{i:x_i \leqslant x} f(x_i)$$

Electronic copy available at: https://ssrn.com/abstract=3307566

Lemma 14.6.2 *The probability mass function $f : \mathbb{R} \to [0,1]$ satisfies*

1. *$f(x) \neq 0$ if and only if x belongs to some countable set $\{x_1, x_2, ..\}$*
2. *$\sum_i f(x_i) = 1$*

Let $x_1, x_2, .., x_N$ be the numerical outcomes of N repetitions of some experiment. The average of these outcomes is

$$m = \frac{1}{N} \sum_i x_i$$

In advance of performing these experiments we can represent their outcomes by a sequence $X_1, X_2, .., X_N$ of random variables, and assume that these variables are discrete with a common mass function f. Then, roughly speaking, for each possible value x, about $Nf(x)$ of the X_i will take that value x. So, the average m is about

$$m \approx \frac{1}{N} \sum_x xNf(x) = \sum_x xf(x)$$

where the summation is over all possible values of the X_i. This average is the expectation or mean value of the underlying distribution with mass function f.

Definition 14.6.9 *The mean value or expectation of X with mass function f is defined to be*

$$E[X] = \sum_{x:f(x)>0} xf(x)$$

whenever this sum is absolutely convergent.

Definition 14.6.10 *The joint mass function $f : \mathbb{R}^2 \to [0,1]$ of X and Y is given by*

$$f(x,y) = P(X = x \text{ and } Y = y)$$

The joint probability mass function of two discrete random variables X, Y is

$$P(X = x \text{ and } Y = y) = P(Y = y|X = x) \cdot P(X = x) = P(X = x|Y = y) \cdot P(Y = y) \tag{14.6.13}$$

where $P(Y = y|X = x)$ is the probability of $Y = y$ given that $X = x$. The generalisation is the joint probability distribution of n discrete random variables $X_1, X_2, ..., X_n$ called the chain rule of probability and given by

$$\begin{aligned} P(X_1 = x_1, ..., X_n = x_n) &= P(X_1 = x_1) \times P(X_2 = x_2|X_1 = x_1) \\ &\times P(X_3 = x_3|X_1 = x_1, X_2 = x_2) \\ &\cdots \\ &\times P(X_n = x_n|X_1 = x_1, X_2 = x_2, ..., X_{n-1} = x_{n-1}) \end{aligned} \tag{14.6.14}$$

Since these are probabilities, we have in the case of n variables

$$\sum_i \sum_j \cdots \sum_k P(X_1 = x_{1i}, X_2 = x_{2j}, ..., X_n = x_{nk}) = 1$$

We write $f_{X,Y}$ when we need to stress the role of X and Y.

Electronic copy available at: https://ssrn.com/abstract=3307566

Lemma 14.6.3 *X and Y are independent if and only if*

$$f_{X,Y}(x,y) = f_X(x)f_Y(y) \text{ for all } x,y \in \mathbb{R}$$

More generally, X and Y are independent if and only if $f_{X,Y}(x,y)$ can be factorised as the product $g(x)h(y)$ of a function of x alone and a function of y alone.

Lemma 14.6.4 *The marginal probability mass function*
If the joint probability mass function (pmf) of discrete random variables X and Y if $f_{X,Y}(x,y)$, then the marginal pmf are given by

$$\begin{aligned} f_X(x) &= P(X=x) = \sum_{y \in Range(Y)} P(X=x, Y=y) = \sum_{y \in Range(Y)} f_{X,Y}(x,y) \\ f_Y(y) &= P(Y=y) = \sum_{x \in Range(X)} P(X=x, Y=y) = \sum_{x \in Range(X)} f_{X,Y}(x,y) \end{aligned}$$

Lemma 14.6.5

$$E[g(X,Y)] = \sum_{x,y} g(x,y) f_{X,Y}(x,y)$$

Definition 14.6.11 *Let X and Y be two discrete random variables. Then the conditional probability mass function of Y given $X = x$ is defined as*

$$f_{Y|X}(y|x) = P(Y=y|X=x) = \frac{P(Y=y, X=x)}{P(X=x)} = \frac{f_{X,Y}(x,y)}{f_X(x)}$$

for $y \in Range(Y)$ and $x \in Range(X)$.

Lemma 14.6.6 *If X and Y are discrete random variables, then for $x \in Range(X)$,*

$$\sum_{y \in Range(Y)} f_{Y|X}(y|x) = 1$$

Similarly, for $y \in Range(Y)$,

$$\sum_{x \in Range(X)} f_{Y|X}(y|x) = 1$$

Proof: From the definition of the conditional probability mass function and the marginal probability mass function, we get

$$\begin{aligned} \sum_{y \in Range(Y)} f_{Y|X}(y|x) &= \sum_{y \in Range(Y)} \frac{f_{X,Y}(x,y)}{f_X(x)} \\ &= \frac{f_X(x)}{f_X(x)} = 1 \end{aligned}$$

***Remark* 14.6.1** *Thus, $\{f_{Y|X}(y|x); y \in Range(Y)\}$ is a probability distribution on $Range(Y)$.*

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 14.6.12 *If k is a positive integer, then the kth moment m_k of X is*

$$m_k = E[X^k]$$

The kth central moment σ_k is

$$\sigma_k = E[(X - m_1)^k]$$

The two moments of most use are $m_1 = E[X]$ and $\sigma_2 = E[(X - E(X))^2]$ called the mean and variance of X.
The following lemma discuss when two mass functions agrees in probability.

Lemma 14.6.7 *If $|f_X(x) - f_Y(y)| = \epsilon$, there exits $f(x, y)$ such that $f(x) = f_X(x)$ and $f(y) = f_Y(y)$ and $f(x = y) = \epsilon$. Then we say that $f_X(x)$ agrees with $f_Y(y)$ with probability ϵ.*

14.6.2.2 Continuous random variables

Definition 14.6.13 *The random variable X is called continuous if its distribution function can be expressed as*

$$F(x) = \int_{-\infty}^{x} f(u)du \, , \, x \in \mathbb{R}$$

for some integrable function $f : \mathbb{R} \to [0, \infty)$.

The expectation of a discrete variable X is $E[X] = \sum_x xP(X = x)$ which is an average of the possible values of X, each value being weighted by its probability. For continuous variables, expectations are defined as integrals.

Definition 14.6.14 *The expectation of a continuous random variable X with density function f is*

$$E[X] = \int_{-\infty}^{\infty} xf(x)dx$$

whenever this integral exists.

We shall allow the existence of $\int g(x)dx$ only if $\int |g(x)|dx < \infty$. Note, the definition of the kth moment m_k in Appendix (14.6.2.1) applies to continuous random variables, but the moments of X may not exist since the integral

$$E[X^k] = \int x^k f(x)dx$$

may not converge.
We are now going to define independence of random variables.

Definition 14.6.15 *Random variables X and Y are independent if their joint distribution function factors into the product of their marginal distribution functions*

$$F_{X,Y}(x, y) = F_X(x)F_Y(y)$$

Theorem 14.6.1 *Suppose X and Y are jointly continuous random variables. X and Y are independent if and only if given any two densities for X and Y their product is the joint density for the pair (X, Y), that is,*

$$f_{X,Y}(x, y) = f_X(x)f_Y(y)$$

Independence requires that the set of points where the joint density is positive must be the Cartesian product of the set of points where the marginal densities are positive, that is, the set of points where $f_{X,Y}(x, y) > 0$ must be (possibly infinite) rectangles.

Electronic copy available at: https://ssrn.com/abstract=3307566

14.6.3 The characteristic function, moments and cumulants

14.6.3.1 Definitions

We start by recalling some definitions together with the properties of the characteristic functions. The characteristic function of a random variable is the Fourier transform of its distribution

Definition 14.6.16 *The characteristic function of a $\mathbb{R}^d$ random variable X is the function $\Phi_X : \mathbb{R}^d \to \mathbb{C}$ defined by*

$$\Phi_X(z) = E[e^{iz.X}] = \int_{\mathbb{R}^d} e^{iz.x} d\mu_X(x) \text{ for } z \in \mathbb{R}^d$$

where μ_X is the measure of X.

The characteristic function of a random variable completely characterises its law. Smoothness properties of Φ_X depend on the existence of moments of the random variable X which is related on how fast the distribution μ_X decays at infinity. If it exists, the n-th moment m_n of a random variable X on $\mathbb{R}$ is

$$m_n = E[X^n]$$

The first moment of X called the mean or expectation measures the central location of the distribution. Denoting the mean of X by μ_X, the nth central moment of X, if it exists, is defined as

$$m_n^c = E[(X - \mu_X)^n]$$

The second central moment σ_X^2 called the variance of X measures the variability of X. The third central moment measures the symmetry of X with respect to its mean, and the fourth central moment measures the tail behaviour of X. In statistics, skewness and kurtosis, respectively the normalised third and fourth central moments of X are used to summarise the extent of asymmetry and tail thickness. The are defined as

$$S = \frac{m_3^c}{(m_2^c)^{\frac{3}{2}}} = E[\frac{(X - \mu_X)^3}{\sigma_X^3}], K = \frac{m_4^c}{(m_2^c)^2} = E[\frac{(X - \mu_X)^4}{\sigma_X^4}]$$

Since $K = 3$ for a normal distribution, the quantity $K - 3$ is called the excess kurtosis. The moments of a random variable are related to the derivatives at 0 of its characteristic function.

***Proposition* 3** *If $E[|X|^n] < \infty$ then Φ_X has n continuous derivatives at $z = 0$ and*

$$m_k = E[X^k] = \frac{1}{i^k} \frac{\partial^k \Phi_X}{\partial z^k}(0)$$

***Proposition* 4** *X possesses finite moments of all orders iff $z \to \Phi_X(z)$ is C^∞ at $z = 0$. Then the moments of X are related to the derivatives of Φ_X by*

$$m_n = E[X^n] = \frac{1}{i^n} \frac{\partial^n \Phi_X}{\partial z^n}(0)$$

If X_i with $i = 1, .., n$ are independent random variables, the characteristic function of $S_n = X_1 + ... + X_n$ is the product of characteristic functions of individual variables X_i

$$\Phi_{S_n}(z) = \prod_{i=1}^{n} \Phi_{X_i}(z) \tag{14.6.15}$$

We see that $\Phi_X(0) = 1$ and that the characteristic function Φ_X is continuous at $z = 0$ and $\Phi_X(z) \neq 0$ in the neighborhood of $z = 0$. It leads to the definition of the cumulant generating function or log characteristic function of X.

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 14.6.17 *There exists a unique continuous function Ψ_X called the cumulant generating function defined around zero such that*

$$\Psi_X(0) = 0 \text{ and } \Phi_X(z) = e^{\Psi_X(z)}$$

The cumulants k_n of a probability distribution are a set of quantities providing an alternative to the moments of the distribution. It is defined via the cumulant-generating function $\Psi_X(z)$, which is the natural logarithm of the moment generating function

$$\Psi_X(z) = \ln \Phi_X(z)$$

The cumulants k_n are obtained from the power series expansion of the cumulant generating function

$$\Psi_X(z) = \sum_{n=1}^{\infty} k_n \frac{z^n}{n!}$$

so that the nth cumulant can be obtained by differentiating the above equation n-times and evaluating the result at zero

$$k_n = \Psi_X^{(n)}(z)\big|_{z=0}$$

14.6.3.2 The first two moments

We consider the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and let X and Y be two discrete random variables. We are going to present some properties of expectations involving joint distributions. For constants $a, b \in \mathbb{R}$, we have

$$E[aX + bY] = aE[X] + bE[Y]$$

Next, given the definition of the kth moment m_k and the kth central moment σ_k in Appendix (14.6.2.1) we get the following definition

Definition 14.6.18

$$\begin{aligned} Cov(X,Y) &= E[XY] - E[X]E[Y] = E\big[(X - E[X])(Y - E[Y])\big] \\ E[XY] &= \sum_{x,y} xy f_{XY}(x,y) \\ Var(X) &= Cov(X,X) = E[X^2] - (E[X])^2 = E\big[(X - E[X])^2\big] \\ \rho(X,Y) &= \frac{Cov(X,Y)}{\big(Var(X)Var(Y)\big)^{\frac{1}{2}}} \text{ whenever } Var(X), Var(Y) \neq 0 \text{ and all these quantities exist} \end{aligned}$$

The covariance measures whether, or not, $(X - E[X])$ and $(Y - E[Y])$ have the same sign. Further, the correlation is scale invariant

$$\rho(aX + b, cY + d) = \rho(X,Y)$$

For random variables X, Y, Z and constants $a, b, c, d \in \mathbb{R}$ then

- $Cov(aX + b, cY + d) = acCov(X,Y)$
- $Cov(X + Y, Z) = Cov(X,Z) + Cov(Y,Z)$
- $Cov(X,Y) = Cov(Y,X)$

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 14.6.2 *For X and Y random variables, whenever the correlation $\rho(X,Y)$ exists, it must satisfy*

$$-1 \leqslant \rho(X,Y) \leqslant 1$$

The correlation $\rho(X,Y)$ is a measure of the strength and direction of the linear relationship between X and Y. Also,

- if X, Y have non-zero variance, then $\rho \in [-1,1]$.
- Y is a linearly increasing function of X if and only if $\rho(X,Y) = 1$.
- Y is a linearly decreasing function of X if and only if $\rho(X,Y) = -1$.

If X and Y are independent, then $\rho(X,Y) = 0$ and we must have $Cov(X,Y) = 0$ which leads to

$$E[XY] = E[X]E[Y]$$

***Remark* 14.6.2** *Independence of X and Y implies $Cov(X,Y) = 0$ but not vice versa. That is, some times $\rho(X,Y) = 0$ and X and Y are dependent.*

Moreover, if X and Y are independent, we get

$$Var(X+Y) = Var(X) + Var(Y)$$

Otherwise, if they are correlated, that is $\rho(X,Y) \neq 0$ we set $Z = X + Y$ and plug it back into the variance equation

$$\begin{aligned} Var(Z) &= E[Z^2] - (E[Z])^2 = E[(X+Y)^2] - (E[X+Y])^2 \\ &= E[X^2 + Y^2 + 2XY] - \big(E[X] + E[Y]\big)^2 \\ &= E[X^2] + E[Y^2] + E[2XY] - (E[X])^2 - (E[Y])^2 - 2E[X]E[Y] \\ &= E[X^2] - (E[X])^2 + E[Y^2] - (E[Y])^2 + 2\big(E[XY] - E[X]E[Y]\big) \\ &= Var(X) + Var(Y) + 2Cov(X,Y) \end{aligned}$$

More generally, for n random variables $X_1, ..., X_n$ the variance becomes

$$Var(\sum_{i=1}^{n} X_i) = \sum_{i=1}^{n} Var(X_i) + \sum_{i \neq j}^{n} Cov(X_i, X_j) \tag{14.6.16}$$

14.6.4 Conditional moments

The related concept of conditional probability dates back from Laplace who calculated conditional distributions. Kolmogorov [1933] formalised it by using the Radon-Nikodym theorem. Halmos and Doob [1953] generalised the conditional expectation by using sub-sigma-algebras.

14.6.4.1 Conditional expectation

Let A and B be two events defined on a probability space. Let $f_n(A)$ denote the number of times A occurs divided by n. As n gets large $f_n(A)$ should be close to $P(A)$, that is, $\lim_{n\to\infty} f_n(A) = P(A)$. When computing $P(A|B)$ we do not want to count the occurrences of $A \cap B^c$ since we know B has occurred. Hence, counting only the occurrences of A where B also occurs, this is $nf_n(A \cap B)$. Now the number of trials is the number of occurrences of B (all other trials are discarded as impossible since B has occurred). Therefore, the number of relevant trials is $nf_n(B)$. Consequently we should have

Electronic copy available at: https://ssrn.com/abstract=3307566

$$P(A|B) \approx \frac{nf_n(A \cap B)}{nf_n(B)} = \frac{f_n(A \cap B)}{f_n(B)}$$

and taking limits in n motivates the definition of the conditional probability which is

Definition 14.6.19 *Given A and B two events, if $P(B) > 0$ then the conditional probability that A occurs given that B occurs is*

$$P(A/B) = \frac{P(A \cap B)}{P(B)}$$

A family $B_1, B_2, ..., B_n$ of events is called a partition of Ω if

$$B_i \cap B_j = \oslash \text{ when } i \neq j \text{ and } \bigcup_{i=1}^{n} B_i = \Omega$$

Lemma 14.6.8 *(Partition Equation) For any events A and B*

$$P(A) = P(A|B)P(B) + P(A|B^c)P(B^c)$$

More generally, let $B_1, B_2, ..., B_n$ be a (finite or countable) partition of Ω. Then

$$P(A) = \sum_{i=1}^{n} P(A|B_i)P(B_i)$$

This may be set in the more general context of the conditional distribution of one variable Y given the value of another variable X (Discrete Case). Let X and Y be two random variables with Y taking values in $\mathbb{R}$ and with X taking only countably many values.

***Remark* 14.6.3** *Suppose we know that the event $\{X = j\}$ for some value j has occurred. The expectation of Y may change given this knowledge.*

Indeed, if $Q(\Lambda) = P(\Lambda|X = j)$, it makes more sense to calculate $E^Q[Y]$ than it does to calculate $E^P[Y]$.

Definition 14.6.20 *Let X have values in $(x_1, x_2, ..., x_n, ..)$ and Y be a random variable. Then if $P(X = x_j) > 0$ the conditional expectation of Y given $\{X = x_j\}$ is defined to be*

$$E[Y|X = x_j] = E^Q[Y]$$

where Q is the probability given by $Q(\Lambda) = P(\Lambda|X = j)$, provided $E^Q[|Y|] < \infty$.

Theorem 14.6.3 *In the previous setting, and if further Y is countably valued with values $(y_1, y_2, ..., y_n, ..)$ and if $P(X = x_j) > 0$, then*

$$E[Y|X = x_j] = \sum_{k=1}^{\infty} y_k P(Y = y_k|X = x_j)$$

provided the series is absolutely convergent.

Electronic copy available at: https://ssrn.com/abstract=3307566

Proof:

$$E[Y|X = x_j] = E^Q[Y] = \sum_{k=1}^{\infty} y_k Q(Y = y_k) = \sum_{k=1}^{\infty} y_k P(Y = y_k|X = x_j)$$

Recall, $P(Y = y_k|X = x_j) = \frac{P(Y=y_k, X=x_j)}{P(X=x_j)}$ so that we get

$$E[Y|X = x_j] = \sum_{k=1}^{\infty} y_k \frac{P(Y = y_k, X = x_j)}{P(X = x_j)}$$

Definition 14.6.21 *Conditional distribution and mass function*
The conditional distribution function of Y given $X = x$, written $F_{Y|X}(.|x)$ is defined by

$$F_{Y|X}(y|x) = P(Y \leqslant y|X = x)$$

for any x such that $P(X = x) > 0$.
The conditional mass function of Y given $X = x$, written $f_{Y|X}(.|x)$ is defined by

$$f_{Y|X}(y|x) = P(Y = y|X = x)$$

As a result, the conditional density function of $Y|X$ can be written as

$$f_{Y|X}(y|x) = \frac{f_{X,Y}(x, y)}{f_X(x)}$$

if $f_X(x) > 0$ and 0 otherwise. $f_{X,Y}(x, y)$ is the joint mass function given in Definition (14.6.10). If X and Y are independent, then $f_{X,Y}(x, y) = f_X(x)f_Y(y)$ and we get $f_{Y|X}(y|x) = f_Y(y)$. Rearranging the above expression, we get

$$f_{X,Y}(x, y) = f_{Y|X}(y|x)f_X(x)$$

and integrating both sides over x we get

$$f_Y(y) = \int_{-\infty}^{\infty} f_{Y|X}(y|x)f_X(x)dx$$

which is an application of the law of total probability for the continuous case.

Next, still with X having at most a countable number of values, we wish to define the conditional expectation of any real valued r.v. Y given knowledge of the random variable X, rather than given only the event $\{X = x_j\}$. To this end we consider the function

$$f(x) = \begin{cases} E[Y|X = x] \text{ if } P(X = x) > 0 \\ \text{any arbitrary value if } P(X = x) = 0 \end{cases}$$

Definition 14.6.22 *Let X be countably valued and let Y be a real valued random variable. The conditional expectation of Y given X is defined to be*

$$E[Y|X] = f(X)$$

where f is given above and provided that it is well defined.

Definition 14.6.23 *Let $\psi(x) = E[Y|X = x]$. Then $\psi(x)$ is called the conditional expectation of Y given X, written $E[Y|X]$.*

Electronic copy available at: https://ssrn.com/abstract=3307566

***Remark* 14.6.4** *Note, a conditional expectation is actually a random variable.*

Theorem 14.6.4 *The conditional expectation* $\psi(X) = E[Y|X]$ *satisfies*

$$E[\psi(X)] = E[Y]$$

Theorem 14.6.5 *Let* $\mathcal{G}, \mathcal{H}$ *be* σ*-algebra such that* $\mathcal{G} \subset \mathcal{H}$*. Then,*

$$E[X|\mathcal{G}] = E[E[X|\mathcal{H}]|\mathcal{G}]$$

Computing expectation by conditioning: Using this theorem, we can compute the marginal mean $E[Y]$ as

$$\begin{aligned} E[Y] &= \sum_x E[Y|X=x]P(X=x) \\ &= \sum_x \sum_y y f_{Y|X}(y|x) f_X(x) \end{aligned} \tag{14.6.17}$$

Thus, the marginal mean is the weighted average of the conditional means, with weights equal to the probability of being in the subgroup determined by the corresponding value of the conditioning variable. Further, from the chain rule of probability in Equation (14.6.14), we have $f_{X,Y}(x,y) = f_{Y|X}(y|x)f_X(x)$, which is a probability function. Hence, we can write the marginal mean as

$$E[Y] = \sum_x \sum_y y f_{X,Y}(x,y) \tag{14.6.18}$$

We can express the variance of the random variable Y in terms of conditional variance as

$$Var(Y) = E[Var(Y|X)] + Var(E[Y|X])$$

In the continuous case, we can compute $E[Y]$ as

$$E[Y] = \int_{-\infty}^{\infty} E[Y|X=x] f_X(x) dx$$

Theorem 14.6.6 *Let* X *and* Y *be random variables, and suppose that* $E[Y^2] < \infty$*. The best predictor of* Y *given* X *is the conditional expectation* $E[Y|X]$*.*

14.6.4.2 Conditional variance

Consider $E[Y|X]$ as a new random variable U as follows: Randomly pick an x from the distribution X so that the new r.v. U has the value $E[Y|X=x]$. For example, $Y = height$ and $X = sex$. Randomly pick a person from the population in question

$$U = \begin{cases} E[Y|X=female] \text{ if the person is female} \\ E[Y|X=male] \text{ if the person is male} \end{cases}$$

If U is a discrete r.v. the expected value of this new random variable is

$$E[U] = \sum_u P(u)u$$

and we get

Electronic copy available at: https://ssrn.com/abstract=3307566

$$E[E[Y|X]] = \text{ weighted average of conditional means } = E[Y]$$

The definition of the (population) (marginal) variance of a random variable Y is

$$Var(Y) = E[(Y - E[Y])^2]$$

Letting $\overline{Y} = E[Y]$ and expending the above term, we can rewrite the variance as

$$Var(Y) = E[Y^2 + (\overline{Y})^2 - 2Y\overline{Y}] = E[Y^2] + (\overline{Y})^2 - 2\overline{Y}E[Y] = E[Y^2] - (\overline{Y})^2$$

Similarly, if we are considering a conditional distribution $Y|X$, we define the conditional variance

$$Var(Y|X) = E\big[(Y - E[Y|X])^2|X\big]$$

Alternatively, we can write the conditional variance as

$$Var(Y|X) = E[Y^2|X] - (E[Y|X])^2 \tag{14.6.19}$$

Proof:
Letting $\psi(X) = E[Y|X]$ we can rewrite the conditional variance as

$$Var(Y|X) = E[Y^2|X] + (\psi(X))^2 - 2\psi(X)E[Y|X] = E[Y^2|X] - (\psi(X))^2$$

As with $E[Y|X]$, we can consider $Var(Y|X)$ as a random variable. Hence, we can compute the expected value of the conditional variance as

$$E[Var(Y|X)] = E[E[Y^2|X]] - E[(E[Y|X])^2]$$

Since the expected value of the conditional expectation of a random variable is the expected value of the original random variable the above equation simplifies to

$$E[Var(Y|X)] = E[Y^2] - E[(E[Y|X])^2] \tag{14.6.20}$$

We can also compute the variance of the conditional expectation as

$$Var(E[Y|X]) = E[(E[Y|X])^2] - (E[E[Y|X]])^2$$

and since $E[E[Y|X]] = E[Y]$ it simplifies to

$$Var(E[Y|X]) = E[(E[Y|X])^2] - (E[Y])^2 \tag{14.6.21}$$

Combining Equation (14.6.20) with Equation (14.6.21) we get

$$E[Var(Y|X)] + Var(E[Y|X]) = E[Y^2] - (E[Y])^2$$

which is the marginal variance $Var(Y)$. That is, the marginal variance is

$$Var(Y) = E[Var(Y|X)] + Var(E[Y|X]) \tag{14.6.22}$$

The marginal (overall) variance is the sum of the expected value of the conditional variance and the variance of the conditional means. Since variances are always non-negative, we get

$$Var(Y) \geqslant E[Var(Y|X)]$$

Further, since $Var(Y|X) \geqslant 0$ and $E[Var(Y|X)]$ must also be positive then

Electronic copy available at: https://ssrn.com/abstract=3307566

$$Var(Y) \geqslant Var(E[Y|X])$$

Note, $E[Var(Y|X)]$ is a weighted average of $Var(Y|X)$. We can also write the variance of the conditional expectation as

$$Var(E[Y|X]) = E\big[(E[Y|X] - E[Y])^2\big]$$

which is a weighted average of $(E[Y|X] - E[Y])^2$.

14.6.5 Multiple random variables

14.6.5.1 Discrete variables

Let $X_1, X_2, ..., X_n$ be an n-dimensional vector of random variables. The joint cumulative distribution function (cdf) is

$$F_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n) = P[X_1 \leqslant x_1, X_2 \leqslant x_2, ..., X_n \leqslant x_n]$$

The joint probability mass function (pmf) of n discrete random variables is

$$f_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n) = P[X_1 = x_1, X_2 = x_2, ..., X_n = x_n]$$

The probability of event A is

$$P(\big(X_1, ..., X_n\big) \in A) = \sum \cdots \sum_{x \in A} f_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n)$$

where $x = (x_1, x_2, ..., x_n)$.
The marginal pmf for X_j is

$$f_{X_j}(x_j) = \sum_{x_1} \cdots \sum_{x_{j-1}} \sum_{x_{j+1}} \cdots \sum_{x_n} f_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n)$$

The marginal pmf for $X_1, X_2, ..., X_{n-1}$ is

$$f_{X_1,X_2,...,X_{n-1}}(x_1, x_2, ..., x_{n-1}) = \sum_{x_n} f_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n)$$

The conditional pmf is

$$f_{X_n}(x_n|x_1, ..., x_{n-1}) = \frac{f_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n)}{f_{X_1,X_2,...,X_{n-1}}(x_1, x_2, ..., x_{n-1})}$$

The chain rule of probability is

$$\begin{aligned} f_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n) &= f_{X_n}(x_n|x_1, ..., x_{n-1}) \\ &\times f_{X_{n-1}}(x_{n-1}|x_1, ..., x_{n-2}) \cdots f_{X_2}(x_2|x_1) f_{X_1}(x_1) \end{aligned}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

14.6.5.2 Continuous variables

The random variables $X_1, X_2, ..., X_n$ are jointly continuous random variables if the probability of any n-dimensional event A is given by an n-dimensional integral of a probability density function

$$P[(X_1, ..., X_n) \in A] = \int \cdots \int_{x \in A} f_{X_1,...,X_n}(x_1^{'}, ..., x_n^{'}) dx_1^{'} ... dx_n^{'}$$

where $f_{X_1,...,X_n}(x_1^{'}, ..., x_n^{'})$ is the joint probability density function (pdf).
The joint cdf of X is obtained from the joint pdf:

$$F_{X_1,...,X_n}(x_1, ..., x_n) = \int_{-\infty}^{x_1} \cdots \int_{-\infty}^{x_n} f_{X_1,...,X_n}(x_1^{'}, ..., x_n^{'}) dx_1^{'} ... dx_n^{'}$$

The joint pdf (if the derivatives exist) is given by

$$f_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n) = \frac{\partial^n}{\partial x_1 \cdots \partial x_n} F_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n)$$

The marginal pdf for a subset of random variables is obtained by integrating the other variables out. For instance, the marginal pdf of X_1 is

$$f_{X_1}(x_1) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f_{X_1,X_2,...,X_n}(x_1, x_2^{'}, ..., x_n^{'}) dx_2^{'} ... dx_n^{'}$$

The marginal pdf for $X_1, ..., X_{n-1}$ is given by

$$f_{X_1,...,X_{n-1}}(x_1, ..., x_{n-1}) = \int_{-\infty}^{\infty} f_{X_1,...,X_n}(x_1, ..., x_{n-1}, x_n^{'}) dx_n^{'}$$

The conditional pdf is

$$f_{X_n}(x_n | x_1, ..., x_{n-1}) = \frac{f_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n)}{f_{X_1,X_2,...,X_{n-1}}(x_1, x_2, ..., x_{n-1})}$$

The chain rule of probability is

$$\begin{aligned} f_{X_1,X_2,...,X_n}(x_1, x_2, ..., x_n) &= f_{X_n}(x_n | x_1, ..., x_{n-1}) \\ &\times f_{X_{n-1}}(x_{n-1} | x_1, ..., x_{n-2}) \cdots f_{X_2}(x_2 | x_1) f_{X_1}(x_1) \end{aligned}$$

The variables $X_1, ..., X_n$ are independent if

$$P[X_1 \in A_1, ..., X_n \in A_n] = P[X_1 \in A_1] \cdots P[X_n \in A_n]$$

The variables $X_1, ..., X_n$ are independent if and only if

$$F_{X_1,...,X_n}(x_1, ..., x_n) = F_{X_1}(x_1) \cdots F_{X_n}(x_n) \, , \forall x_1, ..., x_n$$

In the case of discrete random variables, the above equation becomes

$$f_{X_1,...,X_n}(x_1, ..., x_n) = f_{X_1}(x_1) \cdots f_{X_n}(x_n) \, , \forall x_1, ..., x_n$$

If Z is a function of several random variables

$$Z = g(X_1, X_2, ..., X_n)$$

Electronic copy available at: https://ssrn.com/abstract=3307566

the cdf of Z is

$$P[Z \leqslant z] = P[R_z = \{x = (x_1, ..., x_n) : g(x) \leqslant z\}]$$

and

$$F_Z(z) = P[X \in R_z] = \int \cdots \int_{x \in R_z} f_{X_1,...,X_n}(x_1^{'}, ..., x_n^{'}) dx_1^{'} ... dx_n^{'}$$

The pdf of Z is obtained by taking the derivative of $F_Z(z)$.

14.6.6 Simple random walk

One of the simplest random processes is the simple random walk. At any time a particle inhabits one of the integer points of the real line. At time 0 it starts from point a, and at each subsequent epoch of time $1, 2, ...$ it moves from its current position to a new one according to a law. With probability p it moves one step to the right and with probability $q = 1 - p$ it moves one step to the left. The moves are independent of each other. The walk is symmetric if $p = q = \frac{1}{2}$. Let S_n be the position of the particle after n moves, and set $S_0 = a$. Then,

$$S_n = a + \sum_{i=1}^{n} X_i$$

where $X_1, X_2, ...$ is a sequence of independent Bernoulli variables taking values $+1$ and -1 with probability p and q. The path of a particle is the collection of points in the plane, joined by solid lines between neighbours. The sequence $\{(n, S_n); n \geqslant 0\}$ has some important properties.

Lemma 14.6.9 *The simple random walk is spatially homogeneous, that is,*

$$P(S_n = j | S_0 = a) = P(S_n = j + b | S_0 = a + b)$$

Lemma 14.6.10 *The simple random walk is temporally homogeneous, that is,*

$$P(S_n = j | S_0 = a) = P(S_{n+m} = j | S_m = a)$$

Lemma 14.6.11 *The simple random walk has the Markov property, that is,*

$$P(S_{n+m} = j | S_0, S_1, ..., S_n) = P(S_{n+m} = j | S_n)$$

Thus, conditional upon knowing the value of the process at the nth step, its values after the nth step do not depend on its values before the nth step.

***Remark* 14.6.5** *Statements such as $P(S = j|X, Y) = P(S = j|X)$ are to be interpreted as meaning $P(S = j|X = x, Y = y) = P(S = j|X = x)$ for all x and y. This is a slight abuse of notation.*

14.6.7 Some limits

Definition 14.6.24 *We say that the sequence $F_1, F_2, ...$ of distribution functions converges to the distribution function F, written $F_n \to F$, if $F(x) = \lim_{n \to \infty} F_n(x)$ at each point x where F is continuous.*

We want the sequences $\{F_n\}$ and $\{G_n\}$ to have the same limit in order to drop the requirement that $F_n(x) \to F(x)$ at the point of discontinuity of F.

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 14.6.7 *Continuity*
Suppose that $F_1, F_2, ...$ is a sequence of distribution functions with corresponding characteristic functions $\phi_1, \phi_2, ...$

1. *if $F_n \to F$ for some distribution function F with characteristic function ϕ, then $\phi_n(t) \to \phi(t)$ for all t.*
2. *conversely, if $\phi(t) = \lim_{n\to\phi_n(t)}$ exists and is continuous at $t = 0$, then ϕ is the characteristic function of some distribution function F, and $F_n \to F$.*

We can now deal with the convergence of sequence of random variables and discuss the law of large numbers and the central limit theorem.

Definition 14.6.25 *If $X_1, X_2, ..., X$ is a collection of random variables with distributions $F_1, F_2, ..., F$, then we say that X_n converges in distribution to X, written $X_n \xrightarrow{D} X$, if $F_n \to F$.*

We can rewrite this definition in terms of random variables as follow:

Theorem 14.6.8 *Law of large numbers*
Let $X_1, X_2, ...$ be a sequence of i.i.d. random variables with finite means μ. Their partial sums $S_n = \sum_{i=1}^n X_i$ satisfy

$$\frac{1}{n}S_n \xrightarrow{D} \mu \text{ as } n \to \infty$$

We showed that for large n, then S_n is about as big as $n\mu$. So long as the X_i have finite variance, then $S_n - n\mu$ is about as big as $\sqrt{n}$.

Theorem 14.6.9 *Central limit theorem*
Let $X_1, X_2, ...$ be a sequence of i.i.d. random variables with finite means μ and finite non-zero variances σ^2. Their partial sums $S_n = \sum_{i=1}^n X_i$ satisfy

$$\frac{S_n - n\mu}{\sqrt{n\sigma^2}} \xrightarrow{D} N(0,1) \text{ as } n \to \infty$$

That is, the distribution of S_n, suitably normalised to have mean 0 and variance 1, converges to the distribution function of $N(0,1)$. We see that $N(0,1)$ is a distribution and not a random variable. One can show that this result also applies at the level of density functions and mass functions, given some condition of smoothness.

Theorem 14.6.10 *Local limit theorem*
Let $X_1, X_2, ...$ be a sequence of i.i.d. random variables with zero mean and unit variance, and suppose that their common characteristic function ϕ satisfies

$$\int_{-\infty}^{\infty} |\phi(t)|^r dt < \infty$$

for some integer $r \geqslant 1$. The density function g_n of $U_n = n^{-\frac{1}{2}} \sum_{i=1}^n X_i$ exists for $n \geqslant r$, and furthermore

$$g_n(x) \to \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \text{ as } n \to \infty$$

uniformly in $x \in \mathbb{R}$.

Electronic copy available at: https://ssrn.com/abstract=3307566

14.6.8 Some Markov properties

Let $\{X_0, X_1, ...\}$ be a sequence of random variables taking values in some countable set S. Each X_n is a discrete random variable (d.r.v.) taking one of N possible values, where $N = |S|$.

Definition 14.6.26 *The process X is a Markov chain if it satisfies the Markov condition*

$$P(X_n = s|X_0 = x_0, X_1 = x_1, ..., X_{n-1} = x_{n-1}) = P(X_n = s|X_{n-1} = x_{n-1}) \tag{14.6.23}$$

for all $n \geqslant 1$ and all $s, x_0, x_1, ..., x_{n-1} \in S$.

This definition assume that both conditional probabilities are well defined, that is, $P(X_0 = x_0, X_1 = x_1, ..., X_{n-1} = x_{n-1}) > 0$.

***Remark* 14.6.6** *Markov chains can be described by a sequence of directed graphs, where the edges of graph n are labelled by the probabilities of going from one state at time n to the other states at time $n + 1$, denoted $P(X_{n+1} = x|X_n = x_{n+1})$. The same information is represented by the transition matrix from time n to time $n + 1$.*

Equivalently, for each $s \in S$ and for every sequence $\{x_i; i \geqslant 0\}$ in S, we have

$$P(X_{n+1} = s|X_{n_1} = x_{n_1}, X_{n_2} = x_{n_2}, ..., X_{n_k} = x_{n_k}) = P(X_{n+1} = s|X_{n_k} = x_{n_k}) \text{ for all } n_1 < n_2 < ... < n_k \leqslant n \tag{14.6.24}$$

and

$$P(X_{n+m} = s|X_0 = x_0, X_1 = x_1, ..., X_n = x_n) = P(X_{n+m} = s|X_n = x_n) \text{ for any } m, n \geqslant 0$$

Since S is a countable set, it can be put in a one-to-one correspondence with some subset $S^{'}$ of the integers. For simplicity of exposition we let S be the set $S^{'}$ of integers. Thus, if $X_n = i$, the chain is in the ith state at the nth step. The evolution of the chain is described by its transition probability

$$P(X_{n+1} = j|X_n = i)$$

which depends the three quantities n, i, j. We will only consider chains depending upon i and j, that is, homogeneous.

Definition 14.6.27 *The chain X is called homogeneous if*

$$P(X_{n+1} = j|X_n = i) = P(X_1 = j|X_0 = i)$$

for all n, i, j. The transition matrix $P = (p_{ij})$ is the $|S| \times |S|$ matrix of transition probabilities

$$p_{ij} = P(X_{n+1} = j|X_n = i)$$

Thus, we let X be a Markov chain with transition matrix P.

Theorem 14.6.11 *P is a stochastic matrix, which is to say that*

1. *P has non-negative entries, or $p_{ij} \geqslant 0$.*
2. *P has row sums equal to one, or $\sum_j p_{ij} = 1$.*

In the short term the random evolution of X is described by P, while in the long term

Electronic copy available at: https://ssrn.com/abstract=3307566

Definition 14.6.28 *The n-step transition matrix $P_n = (p_{ij}(n))$ is the matrix of n-step transition probabilities*

$$p_{ij}(n) = P(X_{m+n} = j | X_m = i)$$

Note, $P_1 = P$.

Theorem 14.6.12 *Chapman-Kolmogorov equation*

$$p_{ij}(m+n) = \sum_k p_{ik}(m) p_{kj}(n)$$

Hence $P_{m+n} = P_m P_n$ and so $P_n = P^n$, the nth power of P.

Proof:

$$\begin{aligned} p_{ij}(m+n) &= P(X_{m+n} = j | X_0 = i) \\ &= \sum_k P(X_{m+n} = j, X_m = k | X_0 = i) \\ &= \sum_k P(X_{m+n} = j | X_m = k, X_0 = i) P(X_m = k | X_0 = i) \\ &= \sum_k P(X_{m+n} = j | X_m = k) P(X_m = k | X_0 = i) \end{aligned}$$

as required. In the third line we have used the following result

$$P(A \cap B | C) = P(A | B \cap C) P(B | C)$$

and, in the fourth line, the Markov property in Equation (14.6.24).
This theorem relates the long-term development to the short-term one by telling us how X_n depends on the initial variable X_0. The rest follows from the definition of matrix multiplication.
We let $\mu_i^{(n)} = P(X_n = i)$ be the mass function (density) of X_n, and write $\mu^{(n)}$ for the row vector with entries $(\mu_i^{(n)}; i \in S)$.

Lemma 14.6.12

$$\mu^{(m+n)} = \mu^{(m)} P_n$$

and hence $\mu^{(n)} = \mu^{(0)} P^n$.

Proof:

$$\begin{aligned} \mu_j^{(m+n)} &= P(X_{m+n} = j) = \sum_i P(X_{m+n} = j | X_m = i) P(X_m = i) \\ &= \sum_i \mu_i^{(m)} p_{ij}(n) = (\mu^{(m)} P_n)_j \end{aligned}$$

and the result follows from Theorem (14.6.12).
Thus, the random evolution of the chain is determined by the transition matrix P and the initial mass function $\mu^{(0)}$. Hence, the study of the chain is largely reducible to the study of algebraic properties of matrices.

Electronic copy available at: https://ssrn.com/abstract=3307566

14.6.9 Ergodic theory

A central concern of ergodic theory is the behaviour of a dynamical system when it is allowed to run for a long time. The first result in this direction is the Poincare recurrence theorem, which claims that almost all points in any subset of the phase space eventually revisit the set. More precise information is provided by various ergodic theorems which assert that, under certain conditions, the time average of a function along the trajectories exists almost everywhere and is related to the space average. Two of the most important theorems are those of Birkhoff [1931] and von Neumann which assert the existence of a time average along each trajectory. For the special class of ergodic systems, this time average is the same for almost all initial points: statistically speaking, the system that evolves for a long time forgets its initial state. Stronger properties, such as mixing and equidistribution, have also been extensively studied.
Ergodic theory is often concerned with ergodic transformation.

Definition 14.6.29 *Let $T : X \to$ be a measure-preserving transformation on a measure space (X, Σ, μ), with $\mu(X) = 1$. Then T is ergodic if for every E in Σ with $T^{-1}(E) = E$, either $\mu(E) = 0$ or $\mu(E) = 1$.*

Theorem 14.6.13 *Let $T : X \to$ be a measure-preserving transformation on a measure space (X, Σ, μ) and suppose f is a μ-integrable function, that is, $f \in L_1(\mu)$. Then we define the following averages:*

- *Time average: This is defined as the average (if it exists) over iterations of T starting from some initial point x*

$$\widehat{f}(x) = \lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} f(T^k x)$$

- *Space average: If $\mu(X)$ is finite and nonzero, we can consider the space or phase average of f*

$$\overline{f} = \frac{1}{\mu(X)} \int f d\mu$$

In general the time average and space average may be different. But if the transformation is ergodic, and the measure is invariant, then the time average is equal to the space average almost everywhere. The equidistribution theorem is a special case of the ergodic theorem, dealing specifically with the distribution of probabilities on the unit interval.
The pointwise or strong ergodic theorem states that the limit in the definition of the time average of f exists for almost every x and that the (almost everywhere defined) limit function $\widehat{f}$ is integrable:

$$\widehat{f} \in L_1(\mu)$$

Further, $\widehat{f}$ id T-invariant, that is

$$\widehat{f} \circ T = \widehat{f}$$

holds almost everywhere, and if $\mu(X)$ is finite, then the normalisation is the same

$$\int \widehat{f} d\mu = \int f d\mu$$

In particular, if T is ergodic, then $\widehat{f}$ must be constant (almost everywhere), and one has

$$\overline{f} = \widehat{f}$$

almost everywhere. Joining the first to the last claim, and assuming that $\mu(X)$ is finite and nonzero, one has that

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} f(T^k x) = \frac{1}{\mu(X)} \int f d\mu$$

Electronic copy available at: https://ssrn.com/abstract=3307566

for almost all x. That is, for all x except for a set of measure zero. For an ergodic transformation, the time average equals the space average almost surely.
We now give a probabilistic formulation due to Birkhoff and Khinchin.

Theorem 14.6.14 *Birkhoff-Khinchin theorem*
Let f be measurable, $E(|f|) < \infty$, and T be a measure-preserving map. Then with probability one:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} f(T^k x) = E[f|\mathcal{C}](x)$$

where $E[f|\mathcal{C}]$ is the conditional expectation given the σ-algebra $\mathcal{C}$ of invariant sets of T.

Corollary 6 *Pointwise ergodic theorem*
In particular, if T is also ergodic, then $\mathcal{C}$ is the trivial σ-algebra, and thus with probability one:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} f(T^k x) = E[f](x)$$

The origins of ergodic theory lie in the mechanics of gases (large-scale effects of the molecular dynamics) where the key rationale is that the systems considered are in equilibrium. It is permissible under strict conditions of stationarity (see Grimmet et al. [1992]). While the literature on ergodic systems is concerned with deterministic dynamics, the basic question whether time averages may be replaced by ensemble averages is equally applicable to stochastic systems, such as Langevin equations or lotteries. The essence of ergodicity is the question whether the system when observed for a sufficiently long time t samples all states in its sample space in such a way that the relative frequencies $f(x,t)dx$ with which they are observed approach a unique (independent of initial conditions) probability $P(x)dx$

$$\lim_{t\to\infty} f(x,t) = P(x)$$

If this distribution does not exist or is not unique, then the time average

$$\overline{A} = \lim_{T\to\infty} \frac{1}{T} \int_0^T A(x(t))dt$$

of an observable A can not be computed as an ensemble average in Huygens' sense,

$$< A >= \int_x A(x)P(x)dx$$

14.7 Introduction to stochastic control

For more details see Fleming et al. [1975], Davis [1993], among others. We follow the notation in Oksendal [1998].

14.7.1 Description of the problem

We let the state of a system be represented by an Ito process X with dynamics

$$dX(t) = dX_u(t) = b(t, X(t), u_t)dt + \sigma(t, X(t), u_t)dW(t)$$

where $X(t) \in \mathbb{R}^n$, $b : \mathbb{R} \times \mathbb{R}^n \times U \to \mathbb{R}^n$, $\sigma : \mathbb{R} \times \mathbb{R}^n \times U \to \mathbb{R}^{n\times n}$ and W is an m-dimensional Brownian motion.

***Remark* 14.7.1** *$u_t \in U \subset \mathbb{R}^k$ is a parameter whose value can be chosen in the Borel set in order to control the process X. Thus, $u_t = u(t,\omega)$ is a stochastic process which we assume is $\mathcal{F}_t$-adapted.*

Electronic copy available at: https://ssrn.com/abstract=3307566

We make suitable assumptions on the functions b and σ for the SDE to admit a solution. We denote that solution $\{X_{s,x}(h); h \geqslant s\}$ and let the probability law of $X(t)$ starting at x for $t = s$ be $Q_{s,x}$ such that

$$Q_{s,x}[X(t_1) \in F_1, ..., X(t_k) \in F_k] = P_0[X_{s,x}(t_1) \in F_1, ..., X_{s,x}(t_k) \in F_k]$$

for $s \leqslant t_i$, $F_i \subset \mathbb{R}^n$, $1 \leqslant i \leqslant k$, $k = 1, 2, ...$
We let the utility function $F : \mathbb{R} \times \mathbb{R}^n \times U \to \mathbb{R}$ and the bequest function $K : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ be given continuous functions, we also let G be a fixed domain in $\mathbb{R} \times \mathbb{R}^n$ and let $\widehat{T}$ be the first exit time after s from G for the process $\{X_{s,x}(r); r \geqslant s\}$, that is,

$$\widehat{T} = \widehat{T}_{s,x}(\omega) = \inf\{r > s; (r, X_{s,x}(r, \omega)) \notin G\} \leqslant \infty$$

To simplify notation, we apply a time shift to u_t and introduce

$$Y(t) = (s + t, X_{s,x}(s + t)) \text{ for } t \geqslant 0 \, , Y(0) = (s, x)$$

so that the above SDE becomes

$$dY(t) = dY_u(t) = b(Y(t), u_t)dt + \sigma(Y(t), u_t)dW(t)$$

The probability law for $Y(t)$ is $Q_{s,x} = Q_y$ and

$$\int_s^{\widehat{T}} F_{u_r}(r, X(r))dr = \int_0^T F_{u_{s+t}}(Y(t))dt$$

where

$$T = \inf\{t > 0; Y(t) \notin G\} = \widehat{T} - s$$

Further,

$$K(\widehat{T}, X(\widehat{T})) = K(Y(T))$$

For $y = (s, x)$, the performance function is given by

$$J_u(y) = E_y[\int_0^T F_{u_t}(Y(t))dt + K(Y(T))I(T < \infty)] \tag{14.7.25}$$

where we assume

$$E_y[\int_0^T F_{u_t}(Y(t))dt + K(Y(T))I(T < \infty)] < \infty \text{ for all } y, u \tag{14.7.26}$$

and where $F_u(z) = F(z, u)$.
The problem is, for each $y \in G$, to find the number $\Phi(y)$ and a control $u^* = u^*(t, \omega) = u^*(y, t, \omega)$ such that

$$\Phi(y) = \sup_{u(t,\omega)} J_u(y) = J_{u^*}(y)$$

where the supremum is taken over all processes $\{u_t\}$ with values in U. If it exists, u^* is called an optimal control and Φ is called the optimal performance or the value function. Some examples are

- deterministic control: $u(t, \omega) = u(t)$, also called open loop control.
- closed loop or feedback control, where the process $\{u_t\}$ is $\mathcal{M}_t$-adapted, where $\mathcal{M}_t$ is the σ-algebra generated by $\{X_u(r); r \leqslant t\}$.

Electronic copy available at: https://ssrn.com/abstract=3307566

- Markov control: $u(t,\omega) = u_0(t, X(t,\omega))$ for some function $u_0 : \mathbb{R}^{n+1} \to U \subset \mathbb{R}^k$. Here u does not depend on the starting point $y = (s,x)$. The value we choose at time t only depends on the state of the system at this time. Also, $X(t)$ becomes an Ito diffusion, and in particular a Markov process. In that setting the control is denoted $u(Y) = u(t, X(t))$.

14.7.2 HJB: the Markov control

14.7.2.1 Description

We will only consider the Markov control. In that case the dynamics of the system become

$$dY(t) = b(Y(t), u(Y(t)))dt + \sigma(Y(t), u(Y(t)))dW(t)$$

For $v \in U$ and $f \in C_0^2(\mathbb{R} \times \mathbb{R}^n)$, the diffusion operator is

$$(\mathcal{L}_v f)(y) = \frac{\partial f}{\partial s}(y) + \sum_{i=1}^n b_i(y,v)\frac{\partial f}{\partial x_i}(y) + \sum_{i,j=1}^n a_{ij}(y,v)\frac{\partial^2 f}{\partial x_i \partial x_j}(y)$$

where $a_{ij} = \frac{1}{2}(\sigma\sigma^\top)_{ij}$, $y = (s,x)$ and $x = (x_1, ..., x_n)$. For each function u, the solution $Y(t) = Y_u(t)$ is an Ito diffusion with generator A given by

$$(Af)(y) = (\mathcal{L}_{u(y)} f)(y) \text{ for } f \in C_0^2(\mathbb{R} \times \mathbb{R}^n)$$

For $v \in U$, we define $F_v(y) = F(y,v)$. We get the first fundamental result in stochastic control theory.

Theorem 14.7.1 *HJB equation (I)*
Define

$$\Phi(y) = \sup\{J_u(y); u = u(Y) \textit{ Markov control }\}$$

Suppose that $\Phi \in C^2(G) \cap C(\overline{G})$ is bounded, $T < \infty$ a.s. Q_y for all $y \in G$ and that an optimal Markov control u^* exists. Suppose ∂G is regular for $Y_{u^*}(t)$. Then

$$\sup_{v \in U}\{F_v(y) + (\mathcal{L}_v\Phi)(y)\} = 0 \textit{ for all } y \in G$$

and

$$\Phi(y) = K(y) \textit{ for all } y \in \partial G$$

The above supremum is obtained if $v = u^*(y)$ where $u^*(y)$ is optimal. That is,

$$F(y, u^*(y)) + (\mathcal{L}_{u^*(y)}\Phi)(y) = 0 \textit{ for all } y \in G$$

***Remark* 14.7.2** *The HJB (I) equation states that if an optimal control u^* exists, then we know that its value v at the point y is a point v where the function*

$$v \to F_v(y) + (\mathcal{L}_v\Phi)(y)\,,\ v \in U$$

attains its maximum, and the latter is 0. Thus, the stochastic control problem is transformed into finding the maximum of a real function in $U \subset \mathbb{R}^k$.

So far HJB (I) only states that is is necessary that $v = u^*(y)$ is the maximum of this function. We also need to know if it is sufficient. In our setting we want to know if $u_0(Y)$ is an optimal control. This is the case under some conditions:

Electronic copy available at: https://ssrn.com/abstract=3307566

Theorem 14.7.2 *HJB (II), a converse of HJB (I)*
Let ϕ be a function in $C^2(G) \cap C(\overline{G})$ such that, for all $v \in U$,

$$F_v(y) + (\mathcal{L}_v\phi)(y) \leqslant 0\ ,\ y \in G$$

with boundary values

$$\lim_{t \to T} \phi(Y(t)) = K(Y(T)) \cdot I(T < \infty)\ a.s.\ Q_y$$

and such that $\{\phi(Y(\tau)); \tau \leqslant T\}$ is uniformly Q_y-integrable for all Markov controls u and all $y \in G$. Then

$$\phi(y) \geqslant J_u(y)$$

for all Markov controls u and all $y \in G$.
Further, if for each $y \in G$ we have found $u_0(y)$ such that

$$F_{u_0(y)} + (\mathcal{L}_{u_0(y)}\phi) = 0$$

then $u_0 = u_0(y)$ is a Markov control such that

$$\phi(y) = J_{u_0(y)}(y)$$

and hence u_0 must be an optimal control and $\phi(y) = \Phi(y)$.

While considering only Markov controls is too restrictive, if some extra conditions are satisfied, one can always obtain as good performance with the latter as with an arbitrary $\mathcal{F}_t$-adapted control.

Theorem 14.7.3 *Let*

$$\Phi_M(y) = \sup\{J_u(y); u = u(Y)\ \textit{Markov control}\ \}$$

and

$$\Phi_a(y) = \sup\{J_u(y); u = u(t,\omega)\ \mathcal{F}_t\ \textit{adapted control}\ \}$$

Suppose there exists an optimal Markov control $u_0 = u_0(Y)$ for the Markov control problem such that all the boundary points of G are regular with respect to $Y_{u_0}(t)$ and that Φ_M is a bounded function in $C^2(G) \cap C(\overline{G})$. Then

$$\Phi_M(y) = \Phi_a(y)\ \textit{for all}\ y \in G$$

14.7.2.2 Some examples

Some examples of stochastic control problems are:

- The linear stochastic regulator problem: Suppose that the state $X(t)$ of a system at time t is given by a linear SDE

  $$dX(t) = \big(H(t)X(t) + M(t)u_t\big)dt + \sigma(t)dW(t)\ ,\ t \geqslant s\ ,\ X(s) = x$$

  and the cost is of the form

  $$J_u(s,x) = E_{s,x}[\int_s^{t_1} \big(X^\top(t)C(t)X(t) + u_t^\top D(t)u_t\big)dt + X^\top(t_1)RX(t_1)]\ ,\ s \leqslant t_1$$

Electronic copy available at: https://ssrn.com/abstract=3307566

where all the coefficients $H(t) \in \mathbb{R}^{n\times n}$, $M(t) \in \mathbb{R}^{n\times k}$, $\sigma(t) \in \mathbb{R}^{n\times m}$, $C(t) \in \mathbb{R}^{n\times n}$, $D(t) \in \mathbb{R}^{k\times k}$ and $R \in \mathbb{R}^{n\times n}$ are t-continuous and deterministic. The problem is to choose the control $u = u(t, X(t)) \in \mathbb{R}^k$ such that it minimises $J_u(s, x)$.

- An optimal portfolio selection problem: We let $X(t)$ denotes the wealth of an agent at time t. The agent has the choice between two different investments, a risk-free asset and a risky one (a stock). At each time the agent can choose the fraction u of his wealth he will invest in the risky asset, thus investing $(1 - u)$ in the risk-free asset. Given the dynamics for the risky asset, we obtain the SDE for the wealth $X(t) = X_u(t)$. Starting with the wealth $X(t) = x > 0$ at time t, allowing for no borrowing, the agent wants to maximise the expected utility of the wealth at some future time $t_1 > t$. Letting N be a utility function, the problem is to find $\Phi(s, x)$ and a Markov control $u^* = u^*(t, X(t))$, $0 \leqslant u^* \leqslant 1$, such that

$$\Phi(s, x) = \sup\{J_u(s, x); u \text{ Markov control } , 0 \leqslant u \leqslant 1\} = J_{u^*}(s, x)$$

where $J_u(s, x) = E_{s,x}[N(X_u(T))]$ where T is the first exit time from the region $G = \{(r, z); r < t_1 , z > 0\}$. Details are given in Appendix (14.8).

14.7.3 Terminal conditions

We now consider the case of constraints on Markov controls $Y_u(t)$, such as terminal condition at time $t = T$. One way to solve this problem is to apply a sort of Lagrange multiplier. The problem is as follows: Find $\Phi(y)$ and $u^*(y)$ such that

$$\Phi(y) = \sup_{u\in\mathcal{K}} J_u(y)$$

where

$$J_u(y) = E^y[\int_0^T F_u(Y_u(t))dt + K(Y_u(T))]$$

and where the supremum is taken over the space $\mathcal{K}$ of all Markov controls $u : \mathbb{R}^{n+1} \to U \subset \mathbb{R}^k$ such that

$$E^y[M_i(Y_u(T))] = 0 , i = 1, 2, ..., l$$

where $M = (M_1, ..., M_l) : \mathbb{R}^{n+1} \to \mathbb{R}^l$ is a given continuous function, and

$$E^y[|M(Y_u(T))|] < \infty \text{ for all } y, u$$

We can obtain a related, but unconstrained problem: For each $\lambda \in \mathbb{R}^l$ and each Markov control u, we define

$$J_{u,\lambda}(y) = E^y[\int_0^T F_u(Y_u(t))dt + K(Y_u(T)) + \lambda \cdot M(Y_u(T))]$$

The problem is to find $\Phi_\lambda(y)$ and $u^*_\lambda(y)$ such that

$$\Phi_\lambda(y) = \sup_u J_{u,\lambda}(y) = J_{u^*_\lambda,\lambda}(y)$$

without terminal conditions.

Theorem 14.7.4 *Suppose that for all $\lambda \in \Lambda \subset \mathbb{R}^l$ we can find $\Phi_\lambda(y)$ and u^*_λ solving the above unconstrained stochastic control problem. Further, suppose that there exists $\lambda_0 \in \Lambda$ such that*

$$E^y[M(Y_{u^*_{\lambda_0}}(T))] = 0$$

Then $\Phi(y) : \Phi_{\lambda_0}(y)$ and $u^* := u^*_{\lambda_0}$ solves the original constrained stochastic control problem.

Electronic copy available at: https://ssrn.com/abstract=3307566

14.8 Portfolio optimisation

Using the results on stochastic control presented in Appendix (14.7), we are going to detail the optimal portfolio selection problem. We refer the readers to Fleming et al. [1975], Dana et al. [1994], among others. We will follow the notation from Dana et al. [1994].

14.8.1 Dynamic programming

14.8.1.1 Defining the problem

We assume a continuous time model with a risk-free asset and d risky assets, that is, $(d+1)$ assets. Their dynamics are

$$\begin{aligned} dS_0(t) &= S_0(t)r(t)dt \ , S_0(0) = 1 \\ dS_i(t) &= S_i(t)\big(b_i(t)dt + \sum_{j=1}^{d} \sigma_{ij} dW_j(t)\big) \end{aligned}$$

where W is a d-dimensional Brownian motion. The processes r, b, σ are measurable, $\mathcal{F}_t$-adapted and bounded. $\mathcal{F}_t$ is the tribe composed of the trajectories of W up to t, $r > 0$ and σ is invertible.
Assuming complete market, the agent has an initial capital x at time 0 and want to maximise the expected value of a utility function on consumption and final wealth. As a constraint he can not have debts, that is, he can not borrow (requires $X \geqslant 0$). The agent builds a portfolio with $\theta_i(t)$ assets at time t, with $0 \leqslant \theta_i$. Here, the quantity of the underlying θ_i corresponds to the stochastic control u_t in Appendix (14.7).
His wealth at time t is

$$X(t) = \sum_{i=0}^{d} \theta_i(t) S_i(t)$$

The strategy consists in choosing between the two different investments. We assume that the strategy (policy) is self-financed and that his consumption in $[0,t]$ is given by $\int_0^t c(s)ds$ where c is a positive process $\mathcal{F}_t$-adapted. The strategy θ finance c, so that the dynamics of the wealth become

$$dX(t) = \sum_{i=0}^{d} \theta_i(t) dS_i(t) - c(t)dt \ , X(0) = x$$

We let $\pi_i = \theta_i(t)S_i(t)$, $i \geqslant 1$, be the portion of wealth invested in the ith asset, and denote $X_{\pi,c}(t)$ the process of wealth associated to $\{\pi_i; 1 \leqslant i \leqslant d\}$ and the consumption c.
Applying Ito's lemma, the dynamics of the agent's wealth are given by

$$\begin{aligned} dX_{\pi,c}(t) &= \big(X_{\pi,c}(t)r(t) - c(t)\big)dt + \sum_{i=1}^{d} \pi_i\big(b_i(t) - r(t)\big)dt + \sum_{i,j} \pi_i(t)\sigma_{ij}(t)dW_j(t) \\ X_{\pi,c}(0) &= 0 \end{aligned}$$

Some conditions on c and π are imposed to get a unique solution. Eventually, $X_{\pi,c}$ is an Ito process. We denote $\pi(t) = \big(\pi_i(t), ..., \pi_d(t)\big)^\top$ the portfolio of risky assets at time t.
We let U_1 and U_2 be two functions on $\mathbb{R}_+$ and $\mathbb{R}$, respectively, satisfying

U1 U is strictly concave, strictly increasing of class C^1.

Electronic copy available at: https://ssrn.com/abstract=3307566

U2 $\lim_{x\to\infty} U^{'}(x) = 0$

For a pair (π, c), we have the performance function

$$J(x;\pi,c) = E[\int_0^T U_1(c(t))dt + U_2(X_{\pi,c}(T))]$$

where E is an expectation under P and $x = X_{\pi,c}(0)$. The agent wants to maximise $J(x;\pi,c)$ under the constraint that his wealth $X_{\pi,c}(t)$ must stay positive.

14.8.1.2 Solving the HJB equation

We assume that r, b, σ are deterministic, and first assume that they are constant. The principle of Dynamic Programming (DP) is as follows: We assume that if we hold the wealth X_α at time α, we know how to optimise wealth between α and T, for all X_α. Thus, we consider the set of all strategies leading to wealth X_α at time α and we select the best one. The principle of DP and all the associated proofs can be found in Appendix (14.8.3).
We introduce an extra parameter to define value at the initial time. Given $\alpha \in]0, T[$, we let $X_{\alpha,x}(t)$ be the wealth of an agent with capital x at time α. To simplify notation we have omitted (π, c). The dynamics are given by

$$\begin{aligned} dX_{\alpha,x}(t) &= \big(X_{\alpha,x}(t)r(t) - c(t)\big)dt + \sum_{i=1}^d \pi_i\big(b_i(t) - r(t)\big)dt + \sum_{i,j}\pi_i(t)\sigma_{ij}(t)dW_j(t) \\ X_{\alpha,x}(\alpha) &= 0 \end{aligned}$$

for $\alpha \leqslant t \leqslant T$.
We denote $\mathcal{A}(\alpha, x)$ the set of admissible pairs (π, c). We want to optimise the performance function

$$J(\alpha,x;\pi,c) = E[\int_\alpha^T U_1(c(t))dt + U_2(X_{\alpha,x}(T))]$$

Following Appendix (14.7), we let V be the value function given by

$$V(\alpha,x) = \sup\{J(\alpha,x;\pi,c); (\pi,c) \in \mathcal{A}(\alpha,x)\}$$

The pair (π^*, c^*) is said to be optimal if it is admissible and if

$$J(\alpha,x;\pi^*,c^*) = V(\alpha,x)$$

We can write the principle of DP as follows: For all (α, x), for all $t \geqslant \alpha$

$$V(\alpha,x) = \sup_{(\pi,c)\in\mathcal{A}(\alpha,x)} E[\int_\alpha^t U_1(c(s))ds + V(t, X_{\alpha,x;\pi,c}(t))]$$

Assuming regularity conditions on V, we obtain this function by solving the Hamilton-Jacobi-Bellman equation (see Theorem (14.7.1)) given by

$$\frac{\partial V}{\partial t} + \sup_{(\pi,c)\in\mathbb{R}_+\times\mathbb{R}^d}\Big\{\big[xr(t) - c + \pi^\top(b(t) - r(t)I)\big]\frac{\partial V}{\partial x} + \frac{1}{2}\|\pi^\top\sigma(t)\|^2\frac{\partial^2 V}{\partial x^2} + U_1(c)\Big\} = 0\,,\, t \in [0,T[\,,\, x \in \mathbb{R}_{++}$$

where I is the identity matrix, with the boundary conditions

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\begin{aligned} V(T,x) &= U_2(x)\,,\, x \geqslant 0 \\ V(t,0) &= 0\,,\, t \in [0,T] \end{aligned}$$

Using the infinitesimal generator $\mathcal{L}$ associated to the diffusion of X, the HJB can be rewritten as

$$\frac{\partial V}{\partial t}(t,x) + \sup_{(\pi,c)} \big(\mathcal{L}V(t,x) + U_1(c)\big) = 0 \tag{14.8.27}$$

If we know that there exists an optimal pair (π^*, c^*) and that the value function V satisfies the HJB equation, then we can estimate that pair.
We let the functions I_i be the reciprocal functions of $U_i^{'}$.

Proposition 5 *We assume that there exists an optimal pair (π^*, c^*) and that the value function V satisfies the HJB equation. Then, the pair is*

$$c^*(t) = I_1\big(\frac{\partial V}{\partial x}(t,x)\big)$$

$$\pi^*(t) = -\big[\sigma\sigma^\top(t)\big]^{-1}\big(b(t) - r(t)I\big)\frac{\partial V}{\partial x}(t,x)\big[\frac{\partial V^2}{\partial x^2}(t,x)\big]^{-1}$$

The pair (π^*, c^*) is in a feedback form since it is estimated at time t as a function of the optimal wealth at that time. Note,

- in general it is hard to solve explicitly the HJB equation and to get the value function.
- theses results generalise to the case where the utility function is time dependent.
- $\frac{\pi(t)}{\|\pi(t)\|}$ is independent from the utility functions.

Further, we can have solutions to the HJB equation which are not the value function of our problem.

Theorem 14.8.1 *Let v be a function of $C^{1,2}(]0,T[\times\mathbb{R}_+, \mathbb{R}_+)$ satisfying Equation (14.8.27) and the boundary conditions*

$$\begin{aligned} v(T,x) &= U_2(x)\,,\, x \geqslant 0 \\ v(t,0) &= 0\,,\, t \in [0,T] \end{aligned}$$

Then $V(t,x) \leqslant v(t,x)$, $0 \leqslant t < T$, $0 \leqslant x < \infty$, where V is the value function.

Note, in the case where the pair is admissible and has a feedback control, then v is the value function and the pair (π^*, c^*) is optimal.
Note, in the special case where $U_1(x) = x^\alpha$, $0 < \alpha < 1$ and $U_2(x) = 0$, and the parameters r, b, σ are constants, we can completely solve the optimisation problem. In the case of more general utility functions it is much harder to infer the value function. One solution is to use an alternative approach leading to the computation of Cauchy equations rather than HJB equations.

14.8.2 The martingale approach

Using techniques of martingale and concavity, we are going to show the existence of an optimal pair $(c^*, X^*(T))$. Still using martingale techniques we can associate to that pair a portfolio π^* such that the pair (c^*, π^*) is admissible.

Electronic copy available at: https://ssrn.com/abstract=3307566

14.8.2.1 Admissible strategies

Note, the constraint $X(t) \geqslant 0$ is infinite-dimensional, making it hard to satisfy. We can find an expression such that the trajectories of X no-longer appear, leading to a uni-dimensional constraint. To do so, we need to consider the risk-neutral probability measure. Given

$$\eta(t) = -[\sigma(t)]^{-1}\big(b(t) - r(t)I\big)$$

where I is the identity matrix, and

$$L_t = e^{\int_0^t \eta(s)dW(s) - \frac{1}{2}\int_0^t \|\eta(s)\|^2 ds}$$

and assuming η bounded, we can use Girsanov's theorem. Given $\mathbb{Q}$ the equivalent probability to P defined on $\mathcal{F}_T$ by $dQ = L_T dP$. Then $\tilde{W}(t) = W(t) - \int_0^t \eta(s)ds$ is a $\mathbb{Q} - \mathcal{F}_t$ Brownian motion. The dynamics of the assets, under $\mathbb{Q}$, satisfy

$$dS_i(t) = S_i(t)\big(r(t)dt + \sum_j \sigma_{ij} d\tilde{W}_j(t)\big)$$

and the dynamics of the wealth satisfy

$$dX_{\pi,c}(t) = \big(X_{\pi,c}(t)r(t) - c(t)\big)dt + \sum_{i,j} \pi_i(t)\sigma_{ij}(t)d\tilde{W}_j(t)\,,\, X_{\pi,c}(0) = x$$

Integrating, we get

$$X_{\pi,c}(t)R(t) = x - \int_0^t c(s)R(s)ds + \int_0^t R(s)\pi^\top(s)\sigma(s)d\tilde{W}(s) \tag{14.8.28}$$

where $R(t)$ is the discounting parameter

$$R(t) = e^{-\int_0^t r(s)ds}$$

Definition 14.8.1 *A pair (π, c) is admissible for an initial wealth x if the process $X_{\pi,c}(t)$ in Equation (14.8.28) is of positive or null values. We denote $\mathcal{A}(x)$ the set of admissible pairs.*

Note, if the pair (π, c) is admissible, the process

$$M_t = x + \int_0^t R(s)\pi^\top(s)\sigma(s)d\tilde{W}(s)$$

is a local Q-martingale equal to $X_{\pi,c}(t)R(t) + \int_0^t c(s)R(s)ds$. It is a positive martingale, hence it is a Q-supermartingale satisfying $E^Q[M_T|\mathcal{F}_0] \leqslant M_0$ so that $E^Q[M_T] \leqslant M_0$.

Proposition 6 *Given an admissible pair (π, c) and the associated final wealth $X_{\pi,c}(T)$. Then*

$$E^Q[X_{\pi,c}(T)R(T) + \int_0^T c(s)R(s)ds] \leqslant x \tag{14.8.29}$$

The converse is true.

Electronic copy available at: https://ssrn.com/abstract=3307566

Proposition 7 *Given a process c (positive adapted and bounded process) and an $\mathcal{F}_T$ measurable positive variable Z such that*

$$E^Q[ZR(T) + \int_0^T c(s)R(s)ds] = x$$

Then there exists a previsible portfolio π such that the pair (π, c) is admissible, with the associated final wealth $X_{\pi,c}(T)$ being equal to Z. The market is complete.

The constraint in Equation (14.8.29), under P, can be written as

$$E^P[L(T)R(T)X(T) + \int_0^T L(s)R(s)c(s)ds] = x$$

14.8.2.2 Existence of an optimal pair

The problem of maximising the performance $J(x;\pi,c)$ on the set of admissible pairs (π,c) is such that the final wealth $X_{\pi,c}(T)$ be positive and the pair $(c, X_{\pi,c}(T))$ satisfies the constraint

$$E^P[\int_0^T L(t)R(t)c(t)dt + L(T)R(T)X_{\pi,c}(T)] \leqslant x$$

We denote the value function as

$$V(x) = \sup\{J(x;\pi,c); (\pi,c) \in \mathcal{A}(x)\}$$

We want to find a pair $(c^*, X^*(T))$ that maximise $E[\int_0^T U_1(c(t))dt + U_2(X(T))]$ and satisfies the above constraint. We denote the performance function as

$$J(x;\pi, X_{\pi,c}(T)) = E^P[\int_0^T U_1(c(t))dt + U_2(X_{\pi,c}(T))]$$

and we say that the pair $(X(T), c)$ is admissible if the above constraint is satisfied.
We need to present some important properties of the utility functions.

Proposition 8 *If U satisfies $U1$ and $U2$, then*

$$U(I(y)) - yI(y) = \max\big(U(c) - cy, c \geqslant 0\big)$$

As discussed in Appendix (14.7.3), one way to build an optimal pair is to use the multipliers of Lagrange by considering the Lagrangian. For $\lambda \in \mathbb{R}_+$, we denote

$$\mathcal{L}(c, X(T); \lambda) = E[\int_0^T U_1(c(t))dt + U_2(X(T))] + \lambda E[x - \big(\int_0^T L(t)R(t)c(t)dt + L(T)R(T)X(T)\big)]$$

We want $\lambda^* \in \mathbb{R}_{++}$ such that $(c^*, X^*(T), \lambda^*)$ is a saddle point of $\mathcal{L}$. That is, for all $(c, X(T), \lambda)$, $(c, X(T))$ satisfies the above constraint and $\lambda \in \mathbb{R}_+$

$$\mathcal{L}(c, X(T); \lambda^*) \leqslant \mathcal{L}(c^*, X^*(T); \lambda^*) \leqslant \mathcal{L}(c^*, X^*(T); \lambda)$$

To satisfy the first inequality we want $(c^*(t), X^*(T))$ such that for all (t,ω), $c^*(t,\omega)$ maximise $U_1(c(t,\omega)) - \lambda^* L(t)R(t)c(t,\omega)$, and for all ω, $X^*(T,\omega)$ maximise $U_2(X(T,\omega)) - \lambda^* L(T)R(T)X(T,\omega)$. We obtain the pair

Electronic copy available at: https://ssrn.com/abstract=3307566

$$c^*(t) = I_1(\lambda^*\zeta(t)) \text{ and } X^*(T) = I_2(\lambda^*\zeta(T))$$

where

$$\zeta(t) = R(t)L(t)$$

and λ^* should be the constraint is saturated, that is,

$$E[\int_0^T \zeta(t)I_1(\lambda^*\zeta(t))dt + \zeta(T)I_2(\lambda^*\zeta(T))] = x$$

We let the readers refer to Dana et al. [1994] for a check of optimality. To show existence of λ^* we need to make another assumption on the utility functions.

U3
$$\begin{cases} LRI_1(\lambda\zeta) \in L_1(\Omega \times [0,T]; dP \times dt) \text{ , } \forall \lambda > 0 \\ L(T)R(T)I_2(\lambda\zeta(T)) \in L_1(\Omega; dP) \text{ , } \forall \lambda > 0 \end{cases}$$

Lemma 14.8.1 *Under the assumptions $U1, 2, 3$ the function $\mathcal{X}$ defined on $\mathbb{R}_+$ by*

$$\mathcal{X}(y) = E[\int_0^T \zeta(t)I_1(y\zeta(t))dt + \zeta(T)I_2(y\zeta(T))]$$

is strictly decreasing on $[0, \overline{y}]$ where $\overline{y} = \inf\{y|\mathcal{X}(y) = 0\}$, continuous and satisfies

$$\lim_{y\to 0} \mathcal{X}(y) = +\infty \text{ and } \lim_{y\to\infty} \mathcal{X}(y) = 0$$

Thus, the function admits a continuous inverse denoted $\mathcal{Y}$.

We are left with defining $\lambda^* = \mathcal{Y}(x)$. To get the value function we replace c^* and $X^*(T)$ by their values

$$c^*(t) = I_1(\mathcal{Y}(x)\zeta(t)) \text{ and } X^*(T) = I_2(\mathcal{Y}(x)\zeta(T))$$

and

$$V(x) = G(\mathcal{Y}(x))$$

where G is defined on $\mathbb{R}_+$ by

$$G(y) = E[\int_0^T U_1[I_1(y\zeta(t))]dt + U_2[I_2(y\zeta(T))]]$$

We make the assumption:

U4 U_i is of class C^2 and $U_i^{''}$ is increasing, for $i = 1, 2$.

Thus, one can show that G and $\mathcal{X}$ are of class C^1 and $G^{'}(y) = y\mathcal{X}^{'}(y)$. They can both be written as

$$E[\int_0^T \zeta(t)h_1(y\zeta(t))dt + \zeta(T)h_2(y\zeta(T))]$$

for well chosen functions h_i. These functions are associated to solutions of equations of parabolic types.

Electronic copy available at: https://ssrn.com/abstract=3307566

14.8.3 Dynamic programming: Some proofs

Dynamic programming (DP) belongs to the theory of stochastic control (see Appendix (14.7)) and applies under very general assumptions. It refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner.

14.8.3.1 The principle of dynamic programming

The main idea is as follows:

R1 if we use a strategy on $[\alpha, t]$ and another one on $[t, T]$, we obtain a strategy on $[\alpha, T]$.

R2 if we have a strategy on $[\alpha, T]$, we can decompose it on a strategy on $[\alpha, t]$ and another one on $[t, T]$.

In terms of mathematical optimisation, dynamic programming usually refers to simplifying a decision by breaking it down into a sequence of decision steps over time. This is done by defining a sequence of value functions $V_1, V_2, ..., V_n$ taking y as an argument representing the state of the system at times i from 1 to n. The definition of $V_n(y)$ is the value obtained in state y at the last time n. The values V_i at earlier times $i = n-1, n-2, ..., 2, 1$ can be found by working backwards, using a recursive relationship called the Bellman equation.
In our setting, $[R1]$ leads to

$$E[\int_\alpha^t U_1(c(s))ds + V(t, X_\alpha(t))] \leqslant V(\alpha, x)$$

since $V(t, X_\alpha(t))$ is the optimal strategy on $[t, T]$ and combining a strategy on $[\alpha, t]$ with another one on $[t, T]$ gives a strategy on $[\alpha, T]$.
The equality

$$V(\alpha, x) = \sup_{(\pi,c)\in\mathcal{A}(\alpha,x)} E[\int_\alpha^t U_1(c(s))ds + V(t, X_{\alpha,x;\pi,c}(t))] \tag{14.8.30}$$

comes from the fact that an optimal strategy on $[\alpha, T]$ gives, on $[t, T]$ an optimal strategy.

14.8.3.2 The HJB equation

We can infer the HJB equation from the principle of DP in Equation (14.8.30). To do so we apply Ito's lemma to $V(t, X_{\alpha,x}(t))$ between α and $\alpha + h$, getting

$$V(\alpha+h, X_{\alpha,x}(\alpha+h)) - V(\alpha, x) = \int_\alpha^{\alpha+h} \frac{\partial V}{\partial t}(s, X(s))ds + \int_\alpha^{\alpha+h} \frac{\partial V}{\partial x}(s, X(s))dX_{\alpha,x}(s) + \frac{1}{2}\int_\alpha^{\alpha+h} \frac{\partial^2 V}{\partial x^2}(s, X(s))\|\pi^\top(s)\|^2 ds$$

If we write the principle of DP as

$$V(\alpha, x) \geqslant E[\int_\alpha^{\alpha+h} U_1(c(s))ds + V(\alpha + h, X_{\alpha,x}(\alpha + h))]$$

then, from the above equation, we get

$$\begin{aligned} 0 \quad \geqslant \quad & E[\int_\alpha^{\alpha+h} \Big(U_1(c(s)) + \frac{\partial V}{\partial t}(s, X(s)) + \big(X_{\alpha,x}(s)r(s) - c(s) + \pi^\top(s)(b(s) - r(s)I)\big)\frac{\partial V}{\partial x}(s, X(s)) \\ + \quad & \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(s, X(s))\|\pi^\top(s)\sigma(s)\|^2\Big)ds + \int_\alpha^{\alpha+h} \pi^\top(s)\sigma(s)\frac{\partial V}{\partial x}(s, X(s))dW(s)] \end{aligned}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

We divide by h and let $h \to 0$, getting

$$0 \geqslant U_1(c(\alpha)) + \frac{\partial V}{\partial t}(\alpha, x) + \big(xr(\alpha) - c(\alpha) + \pi^\top(\alpha)(b(\alpha) - r(\alpha)I)\big)\frac{\partial V}{\partial x}(\alpha, x) + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(\alpha, x)\|\pi^\top(\alpha)\sigma(\alpha)\|^2$$

Since this relation is true for all π, c, we get

$$0 \geqslant \frac{\partial V}{\partial t}(\alpha, x) + \sup_{\pi,c}\Big\{U_1(c) + \big(xr(\alpha) - c + \pi^\top(\alpha)(b(\alpha) - r(\alpha)I)\big)\frac{\partial V}{\partial x}(\alpha, x) + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(\alpha, x)\|\pi^\top(\alpha)\sigma(\alpha)\|^2\Big\}$$

and we obtain the HJB equation. In the case where there exists an optimal pair, the above inequalities become equalities.

14.8.3.3 Sufficient conditions for an optimal solution

Assuming constant parameters and $U_2 = 0$, we state the conditions under which the value function is solution to the HJB. We write $U = U_1$ and let the diffusion operator $\mathcal{L}$ satisfy

$$\mathcal{L}H(t, y) = -\frac{\partial H}{\partial t}(t, y) + ry\frac{\partial H}{\partial y}(t, y) - \frac{1}{2}\|\eta\|^2 y^2 \frac{\partial^2 H}{\partial y^2}(t, y)$$

where $\eta = -\sigma^{-1}(b - rI)$. We assume that there exists functions G and S in $C^{1,3}([0, T] \times \mathbb{R}_{++}, \mathbb{R}_+)$ such that

$$\begin{cases} \mathcal{L}G(t, y) = U(I(y)) \, , t \in [0, T] \, , y \in \mathbb{R}_{++} \\ G(T, y) = 0 \, , y \in \mathbb{R}_{++} \end{cases}$$

and

$$\begin{cases} \mathcal{L}S(t, y) = yI(y) \, , t \in [0, T] \, , y \in \mathbb{R}_{++} \\ S(T, y) = 0 \, , y \in \mathbb{R}_{++} \end{cases}$$

We assume that G, $\frac{\partial G}{\partial y}$, S and $\frac{\partial S}{\partial y}$ satisfy the condition of polynomial growth

$$\max_{0 \leqslant t \leqslant T} H(t, y) \leqslant M\big(1 + y^{-\lambda} + y^{\lambda}\big) \, , y \in \mathbb{R}_{++}$$

where M and λ are strictly positive constants. We also assume that U is of class C^2. One can show the following result:

Theorem 14.8.2 *Under the previous assumptions the value function is of class $C^{1,2}([0, T] \times \mathbb{R}_{++})$ and satisfies the boundary conditions.*

Electronic copy available at: https://ssrn.com/abstract=3307566

Chapter 15

Monte Carlo

We briefly describe stochastic differential equations (SDEs) and their discretisation in view of computing expectation with Monte Carlo engine. These SDEs can represent the dynamics of the state of the system. We use notation from option pricing theory since the option price is a particular objective function. Note, the hedge of an option (delta) can be seen as the gradient of the objective function.

See textbooks by Oksendal [1998] and Glasserman [2004].

15.1 The dynamics of the state of the system

15.1.1 The stochastic differential equation

We let the state process $(X_t)_{t\geqslant 0}$ be a d-dimensional Ito process valued in the open subset D with dynamics being

$$dX_t = b(X,t)dt + \sigma(X,t)dW_X(t) \tag{15.1.1}$$

where the drift $b : D \to \mathbb{R}^d$ as well as the diffusion $\sigma^\top : D \to \mathbb{R}^{d\times p}$ are regular enough to have a unique strong solution valued in D. The value of the objective $C(t,x)$ on $]0,T]\times]0,+\infty[$ is

$$C(t,X_t) = E\big[e^{-\int_t^T r_s ds} h(X_T)|\mathcal{F}_t\big] \tag{15.1.2}$$

where h is a sufficiently smooth payoff function. Furthermore, we assume that these value functions are of class $\mathcal{C}^2$ such that for $T > t$ the law of X_T knowing $X_t = x$ has a density $\phi(T,K)$ given by

$$\phi(T,K) = E_t\big[\delta(X_T - K)\big] = \frac{1}{P(t,T)}\partial_{KK}C(t,X_t;K,T)$$

where $P(t,T) = E\big[e^{-\int_t^T r_s ds}\big]$ is a zero-coupon bond (expected reward) and $\delta(\cdot)$ is the Dirac function. The density follows the Fokker-Planck equation

$$\partial_t\phi(x,t) = -\sum_{i=1}^d \partial_{x_i}\Big(b_i(x,t)\phi(x,t) + \alpha\sum_{k=1}^d\sum_{j=1}^p \partial_{x_k}\sigma_{ij}(x,t)\sigma_{kj}(x,t)\phi(x,t)\Big) + \frac{1}{2}\sum_{i=1}^d\sum_{k=1}^d\sum_{j=1}^p \partial_{x_i x_k}\sigma_{ij}(x,t)\sigma_{kj}(x,t)\phi(x,t)$$

If the dynamics are deterministic with $\sigma = 0$, we get the Liouville equation

$$\partial_t\phi(x,t) = -\sum_{i=1}^d \partial_{x_i} b_i(x,t)\phi(x,t)$$

Electronic copy available at: https://ssrn.com/abstract=3307566

Thus, if the dynamics are known but the probability distribution of the initial conditions has finite width, the cloud of possible trajectories is described by the appropriate Liouville equation. The α term corresponds to the Stratonovich systems, while for an Ito system we get $\alpha = 0$. The important thing about the noise induced drift is that it can cause the average dynamics of a system to differ from those of its deterministic counterpart, even though the white noise itself has zero mean. If the diffusion term $\sigma(x,t)$ is independent from x both Stratonovich and Ito systems yield equivalent dynamics.

15.1.2 From Stratonovich equation to Ito equation

Continuous systems, where rapidly varying quantities with small but finite correlation times are treated as white noise, follow Stratonovich calculus. For example, we can consider the dynamics

$$\frac{dx}{dt} = b(x,t) + \sigma(x,t)\xi_0(t)$$

where $\xi_0(t)$ is the rapidly varying quantity we wish to approximate as white noise. One of the restriction on the processes is that their correlation functions must decay rapidly enough in comparison with the timescales of the deterministic functions $b(x,t)$ and $\sigma(x,t)$. The constraints should be considered in a weaker sense. Since $\xi_0(t)$ varies more rapidly than $b(x)$ and $\sigma(x)$ it can be scalled with ϵ. One need the scaled time $\tau = \frac{t}{\epsilon^2}$ such that

$$\epsilon\xi_0(\frac{t}{\epsilon^2}) = \xi(\tau)$$

Assuming $\xi(\tau)$ to be a stationary Markov process with $E[\xi(\tau)] = 0$ and $E[\xi(\tau)\xi(0)] = C(\tau)$ we get

$$E[\xi_0(t)] = 0 \text{ and } E[\xi_0(t)\xi_0(0)] = \frac{1}{\epsilon^2}C(\frac{t}{\epsilon^2})$$

To get the white noise limit we let $\epsilon \to 0$ so that $E[\xi_0(t)\xi_0(0)] \to \delta(t)$. Since $\xi(\tau)$ is a stationary Markov process, its probability distribution function (pdf) obeys a Fokker-Planck equation in the scaled coordinates. Hence, we can let $\epsilon \to 0$ and get the Fokker-Planck equation in the Stratonovich sense

$$\partial_t\phi(x) = -\partial_x b(x)\phi(x) + \frac{1}{2}\partial_x\big(\sigma(x)\partial_x(\sigma(x)\phi(x))\big)$$

corresponding to the stochastic didderential equation (SDE)

$$dx = b(x)dt + \sigma(x).dW$$

Although the limiting dynamics are obtained in the limit of vanishing ϵ, we have no power over the geophysical system we wish to describe. Therefore, we may only estimate what a realistic value of ϵ would be in that system and decide whether it is small enough for the system to be usefully described by its limiting dynamics. Stratonovich calculus results when a continuous system is subjected to the white noise limit.
There do exist physical systems where the appropriate starting point is a discrete relation

$$x(t_{i+1}) = x(t_i) + b(x(t_i))\Delta t + \sigma(x(t_i))\eta(t_i)$$

where $t_{i+1} = t_i + \Delta t$ for all i and $\eta(t_i)$ are independent Gaussian random variables with $E[\eta(t_i)] = 0$ and $Var(\eta(t_i)) = \Delta t$. If the timescales of interest are large compared with Δt we may take the continuous limit getting the SDE

$$dx = b(x)dt + \sigma(x)dW$$

The Euler scheme is a first order Runge-Kutta method used for Ito equations. Integration from the nth time step to the $(n+1)$ time step is

Electronic copy available at: https://ssrn.com/abstract=3307566

$$x_{n+1} = x_n + b(x_n, t)\Delta + \sigma(x_n, t)(\Delta W)_n$$

If R is a vector of centered Gaussian random numbers with autocovariance matrix being the identity matrix I, then $(\Delta W)_n = R\sqrt{\Delta}$. The Euler scheme converges to an Ito process with $o(\Delta^{\frac{1}{2}})$ accuracy. If $\sigma = \sigma(t)$ is independent from x we may use a Euler scheme for a Stratonovich system since the moments of Ito and Stratonovich additive noise processes are the same. That is, one may generate an intermediate Ito variable y and then estimate the desired Stratonovich variable x with

$$x(t) = \big(y(t + \frac{1}{2}\Delta) + y(t - \frac{1}{2}\Delta)\big)$$

In the Heun scheme, a second order Runge-Kutta method, we generate at each time step the Wiener process increment $(\Delta W)_n$ and uptate the equation as

$$x_{n+1} = x_n + \frac{1}{2}\big(b(x_n, t_n) + \mu(x_{n+1}^{'}, t_{n+1})\big)\Delta + \frac{1}{2}\big(\sigma(x_n, t_n) + \sigma(x_{n+1}^{'}, t_{n+1})\big)(\Delta W)_n$$

with the Euler predictor

$$x_{n+1}^{'} = x_n + b(x_n, t)\Delta + \sigma(x_n, t)(\Delta W)_n$$

Unfortunately, the Heun method does not converge uniformly to the correct solution with the time step. There is a transformation from every Ito equation to a Stratonovich equation and vice versa, at least in distribution. It consists of adding the term

$$\frac{1}{2}\sum_{k=1}^{d}\sum_{j=1}^{p}\partial_{x_k}\sigma_{ij}(x,t)\sigma_{kj}(x,t)$$

to the drift term $\mu_i(x,t)$ in a Stratonovich SDE to get the corresponding Ito equation. Similarly, one can substract that term from an Ito SDE to get the corresponding Stratonovich equation. So, one can use the higher accuracy of the Heun method for an Ito equation by adjusting the drift term accordingly. This transformation is only practical for systems where $\sigma(x,t)$ is explicitly known and fairly easy to manipulate. See for example the Milshtein schemes. The question of how large to make the time step naturally arises. It has been found that dividing the smallest physical timescale by 2^6 or 2^7 yields acceptable accuracy.

15.1.3 Discretisation of the SDE

In Section (15.1.2) we briefly discussed methods needed when we simulate trajectories of a given process. We are now going to give more details on the Euler scheem and explain the reasons why it is sufficient when applied in finance. This is needed when we want to compute, using Monte Carlo simulations, a path-dependent option or an option on multi-underlyings. We want to find a scheem approximating the solution of a stochastic differential equation (sde). For simplicity, we consider the process $(X)_{t\geqslant 0}$ given in Equation (15.1.1) and assume stationarity. Hence, its solution becomes

$$X_t = X_0 + \int_0^t b(X_s)ds + \int_0^t \sigma(X_s)dW_s \tag{15.1.3}$$

where $(W)_{t\geqslant 0}$ is a p-dimensional Brownian motion.

Electronic copy available at: https://ssrn.com/abstract=3307566

15.1.3.1 The Euler scheem

We let $h = \frac{T}{m}$ be the discretisation step of the time interval $[0, T]$ and get the exact solution

$$X_h = X_0 + \int_0^h b(X_s)ds + \int_0^h \sigma(X_s)dW_s$$

so that a natural approximation of X_h is

$$X_h \approx X_0 + b(X_0)h + \sigma(X_0)(W_h - W_0)$$

Starting with $\hat{X}_0 = X_0$ and proceeding by induction, we get

$$\hat{X}_{(p+1)h} = \hat{X}_{ph} + b(\hat{X}_{ph})h + \sigma(\hat{X}_{ph})(W_{(p+1)h} - W_{ph})$$

The weak convergence states that if both b and σ are Lipsichtz functions and f is a C^4 function with bounded derivatives up to order 4, then there exists a constant C_T such that

$$|E[f(X_T)] - E[f(\hat{X}_T)]| \leqslant \frac{C_T(f)}{m}$$

The speed of convergence in L^2 is of order $h^{\frac{1}{2}}$, while for very regular functions, the speed of convergence in law of the scheem is of order h.

15.1.3.2 Justification of the Euler scheme

Various scheems of higher order have been proposed for discretising SDEs but they are often almost impossible to implement. The simplest of these scheems is the Milshtein scheem (see Milshtein [**?**]) but it is hard to simulate when more than one Brownian motion is involved and the speed in law remains of order h. That is, the Milshtein scheem improves the trajectorial speed of convergence as it is (almost) of order h compared to $\sqrt{(h)}$ for the Euler scheme. Nevertheless, the convergence speed in law for regular function is of order h in both cases, and it is the speed in law which is relevant to evaluate the performance of Monte Carlo algorithm for option pricing. In addition, the simulation of the Milshtein scheem requires the computation at each time step of all the quantities

$$\partial\sigma_j(X_{jh})\sigma_k(X_{kh})$$

making the scheme much slower than the Euler scheme. A result due to Clark and Cameron proves that in L^2-norm, the Euler scheme is in general almost optimal among the schemes involving only the random variables W_{kh} for $k \geqslant 0$. If we accept to simulate more than the increments of the Brownian motions, one can construct schemes with arbitrary high order of convergence. However, these schemes do not have much practical interest as they require to simulate iterated integrals of the Brownian motion of high order and these integrals are known to be challenging to simulate.

15.1.4 Generating trajectories

15.1.4.1 Multivariate normals

A multivariate normal distribution $N(\mu, \Sigma)$ is specified by its mean vector μ and covariance matrix Σ. The covariance matrix may be specified implicitly by its diagonal entries σ_i^2 and correlation ρ_{ij} and becomes in matrix form $\Sigma = \sigma R \sigma^\top$ where R is the correlation matrix

$$\Sigma = \begin{bmatrix} \sigma_1 & ... & 0 \\ ... & ... & ... \\ 0 & ... & \sigma_n \end{bmatrix} \begin{bmatrix} \rho_{11} & ... & \rho_{1n} \\ ... & ... & ... \\ \rho_{n1} & ... & \rho_{nn} \end{bmatrix} \begin{bmatrix} \sigma_1 & ... & 0 \\ ... & ... & ... \\ 0 & ... & \sigma_n \end{bmatrix}$$

Electronic copy available at: https://ssrn.com/abstract=3307566

From the Linear Transformation property, if $y \sim N(0, I)$ and $X = \mu + Ay$ then $X \sim N(\mu, AA^\top)$. The problem of sampling X from the multivariate normal $N(\mu, \Sigma)$ reduces to finding a matrix A for which $AA^\top = \Sigma$. One approach is to use the Cholesky Factorisation on the covariance matrix Σ. An alternative approach is to use the Cholesky Factorisation on the correlation matrix R obtaining the new matrix $\overline{R}$ such that $\overline{R}\overline{R}^\top = R$. In that case we recover the matrix A from $A = \overline{R}\sigma$.

15.1.4.2 One-dimension

In order to perform simulation of an approximate trajectory of a Brownian motion (BM) we let h be a time discretisation step. We simulate a realisation of the family of independent Gaussian random variables with mean 0 and variance 1 notted y_k for $0 \leqslant k \leqslant m$. To obtain a realisation of W_{ph} for $1 \leqslant p \leqslant m$ we compute the recursion

$$W_{ph} = \sqrt{h} \sum_{1 \leqslant k \leqslant p} y_k \tag{15.1.4}$$

More precisely, we are now going to simulate the values $\big(W(t_1), .., W(t_n)\big)$ at the fixed set points $0 < t_1 < ... < t_n$. To do so we use the fact that Brownian motion has independent normally distributed increments. We let $y_1, .., y_n$ be independent standard normal random variables. Given $t_0 = 0$ and $W(0) = 0$, subsequent values follow from

$$W(t_{i+1}) = W(t_i) + \sqrt{t_{i+1} - t_i} y_{i+1} \text{ for } i = 0, .., n-1$$

giving the increment $W(t_{i+1}) - W(t_i) = \sqrt{t_{i+1} - t_i} y_{i+1}$. From the properties of the Brownian motion we have

$$W(t_{i+1}) = \sum_{j=0}^{i-1} W(t_{j+1}) - W(t_j) = \sum_{j=0}^{i-1} \sqrt{t_{j+1} - t_j} y_{j+1}$$

For $X \sim BM(\mu, \sigma^2)$ with time-dependent coefficients, the recursion becomes

$$X(t_{i+1}) = X(t_i) + \int_{t_i}^{t_{i+1}} \mu(s) ds + \sqrt{\int_{t_i}^{t_{i+1}} \sigma^2(s) ds} y_{i+1} \text{ for } i = 0, .., n-1$$

The joint distribution of the corresponding Brownian motion are matched at times $t_1, .., t_n$ but it says nothing when time is between t_i and t_{i+1}. Using the Euler approximation, the recursion becomes

$$X(t_{i+1}) = X(t_i) + \mu(t_i)(t_{i+1} - t_i) + \sigma(t_i)\sqrt{(t_{i+1} - t_i)} y_{i+1} \text{ for } i = 0, .., n-1$$

introducing discretisation error even at times $t_1, .., t_n$ because the increments no longer have exactly the right mean and variance. In general we need to find the mean vector and covariance matrix of $\big(W(t_1), .., W(t_n)\big)$. For a standard Brownian motion $E[W(t_i)] = 0$ and $Cov(W(s), W(t)) = s$ when $0 < s < t$ so that the covariance is $\Sigma = \min(t_i, t_j)$ and the distribution becomes $N(0, \Sigma)$.

15.1.4.3 Multi-dimensions

In multiple dimension we let $y_1, .., y_n$ be independent $N(0, I)$ random vectors in $\mathbb{R}^d$ where d is the dimension and $W(.)$ is now a standard d-dimensional Brownian motion. To simulate $X \sim BM(\mu, \Sigma)$ we find the matrix A for which $AA^\top = \Sigma$ and use the recursion

$$X(t_{i+1}) = X(t_i) + \int_{t_i}^{t_{i+1}} \mu(s) ds + A(t_i, t_{i+1}) y_{i+1} \text{ for } i = 0, .., n-1$$

with

Electronic copy available at: https://ssrn.com/abstract=3307566

$$A(t_i, t_{i+1})A(t_i, t_{i+1})^\top = \int_{t_i}^{t_{i+1}} \Sigma(s)ds$$

thus requiring n factorisation.

15.1.5 Construction of the Brownian bridge

15.1.5.1 Construction

We saw in Section (15.1.4.2) that one can generate the vector $(W(t_1), .., W(t_n))$ from left to right by using the recursion

$$W(t_{i+1}) = W(t_i) + \sqrt{t_{i+1} - t_i}Z_{i+1}\ , i = 0, .., n-1$$

where Z_i for $i = 1, .., n$ are independent standard normal random variables. However, we can generate the $W(t_i)$ in any order we choose as long as we sample from the correct conditional distribution at each time step given the values already generated. Conditioning a Brownian motion on its endpoints produces a Brownian bridge. Hence, we first determine $W(t_n)$ and then fill in the intermediate values by simulating a Brownian bridge from $0 = W(0)$ to $W(t_n)$. Let's assume that the Brownian path at time $s_1 < s_2 < ... < s_k$ have been determined and that we want to draw $W(s)$ conditional on these values such that $s_i < s < s_{i+1}$. From the Markov property of Brownian motion we get

$$\big(W(s)|W(s_1) = x_1, ..., W(s_k) = x_k\big) = N\big(m(s_i, s_{i+1}; s), v(s_i, s_{i+1}; s)\big)$$

with

$$m(s_i, s_{i+1}; s) = \frac{(s_{i+1} - s)x_i + (s - s_i)x_{i+1}}{s_{i+1} - s_i}\ , v(s_i, s_{i+1}; s) = \frac{(s_{i+1} - s)(s - s_i)}{s_{i+1} - s_i}$$

The conditional mean of $W(s)$ lies on the line segment connecting (s_i, x_i) and (s_{i+1}, x_{i+1}) so that $W(s)$ is normally distributed about this mean with a variance that depends on $(s - s_i)$ and $(s_{i+1} - s)$. Hence, sampling $W(s)$ from this conditional distribution, we get

$$W(s) = m(s_i, s_{i+1}; s) + \sqrt{v(s_i, s_{i+1}; s)}Z$$

with $Z \sim N(0, 1)$ independent of all $W(s_1), .., W(s_k)$. Starting by sampling $W(t_n)$ fom $N(0, t_n)$ we proceed by conditionally sampling intermediate values on the two closest time points already sampled at each step. If n is not a power of 2, one can still apply the algorithm to a subset of $2^m < n$ of the t_i with the remaining points filled in at the end. In the case where the Brownian motion has a drift μ, only sampling the rightmost point would change by sampling W_{t_m} from $N(\mu t_m, t_m)$. This is because the conditional distribution of $W(t_1), .., W(t_{n-1})$ given W_{t_m} is the same for all values of μ. Further, given a Brownian motion with a diffusion coefficient σ^2, the conditional mean is unchanged but the conditional variance is multiplied by σ^2. In the case of a d-dimensional Brownian motion we simply apply independent one-dimensional constructions to each of the coordinates. To include a drift vector it suffices to add $\mu_i t_n$ to $W_i(t_n)$ at the first step of the construction of the ith coordinate. Further, to construct $X \sim BM(\mu, \Sigma)$ we use the fact that X can be represented linearly as $X(t) = \mu t + BW(t)$ with B a (d, k) matrix for $k \leqslant d$ satisfying $BB^\top = \Sigma$ and W is a standard k-dimensional Brownian motion. We then apply a Brownian bridge construction to sample $W(t_1), .., W(t_n)$ and recover $X(t_1), .., X(t_n)$ with a linear transformation.

15.1.5.2 Justification

In general, the Brownian bridge construction has no computational advantage over the simple random walk recursion, and they both share the same distribution. However, the Brownian bridge provides us with a greater control over the coarse structure of the simulated Brownian path. It uses a single normal random variable $W(t_n) = \sqrt{t_n}y_1$ to determine

Electronic copy available at: https://ssrn.com/abstract=3307566

the endpoint of a path which may be the most important feature of the path as opposed to the combined result of n independent normal variables in the simple random walk recursion (see Equation (15.1.4)). That is, the latter proceeds by evolving the path forward through time while the former proceeds by adding increasingly fine detail to the path at each step. According to some authors such as Glasserman [2004] it is relevant to change the order in which coordinates of a sequence are assigned to arguments of an integrand when some coordinates of a low-discrepancy sequence exhibit better uniformity properties than others. This is the case for Sobol' sequences in which coordinates generated by lower-degree polynomials are preferable. That is, all coordinates of a Sobol sequence are not equally well distributed. A general strategy for improving QMC approximation is to apply a change of variables to produce an integrand for which only a small number of arguments are important and then applies the lowest-indexed coordinates of a QMC sequence to those coordinates. For instance, one can combine Sobol' sequences with a Brownian bridge. Normal procedure is for the ith coordinate of each point to be transformed to a sample from the standard normal distribution, and then scaled and summed using the random walk construction. Given that the initial coordinates of a Sobol sequence have uniformity superior to that of higher-indexed coordinates, this construction does a good job of sampling the first few increments of the Brownian path. However, many option contracts would be primarily sensitive to the terminal value of the Brownian path. On the other hand, when using the Brownian bridge construction, the first coordinate of a Sobol sequence determines the terminal value of the Brownian path, and the value becomes particularly well distributed. In addition, the first several coordinates of the Sobol sequence determine the general shape of the Brownian path, while the last few coordinates influence only the fine detail of the path. Put another way, the initial coordinates of a Sobol sequence are more uniform than the later ones such that a strategic assignment of coordinates to sources of randomness can improve accuracy. Note, neither the Brownian bridge nor the principal components construction is tailored to a particular type of option payoff and one should use additional information about the payoff to find the appropriate change of variables.

15.1.6 Simulating the state process

The state value at time T with stochastic volatility is

$$S_T = S_0 C_T e^{-\frac{1}{2}\int_0^T \sigma_s^2 ds + \int_0^T \sigma_s dW_s}$$

We are going to discretise the maturity into n time-steps such that $T = n\Delta t$ so that the state value price rewrite

$$S_T = S_0 C_T e^{-\frac{1}{2}\sum_{j=0}^{n-1}\int_{t_j}^{t_{j+1}} \sigma_j^2 ds + \sum_{j=0}^{n-1}\int_{t_j}^{t_{j+1}} \sigma_j dW_j}$$

Hence, the discretised state value at time t_i becomes

$$S_{t_i} = S_0 C_{t_i} e^{-\frac{1}{2}\sum_{j=0}^{i-1}\int_{t_j}^{t_{j+1}} \sigma_s^2 ds + \sum_{j=0}^{i-1}\sqrt{\frac{1}{\Delta t}\int_{t_j}^{t_{j+1}} \sigma_j^2 ds}\hat{W}_{t_j}}$$

where $\hat{W}_{t_j} = \Delta W_j$ and $\Delta W_j = W(t_j) - W(t_{j-1})$. Recall, to compute the covariance matrix we use the matrix σ

$$\sigma = \begin{bmatrix} \sigma_1^2 & ... & 0 \\ ... & ... & ... \\ 0 & ... & \sigma_n^2 \end{bmatrix}$$

where $\sigma_i^2 = \int_{t_{i-1}}^{t_i} \sigma_s^2 ds$ for $1 \leqslant i \leqslant n$. To simulate the standard Gaussian random variable, we draw the random variable y_i from the identity matrix

$$I = \begin{bmatrix} 1 & ... & 0 \\ ... & ... & ... \\ 0 & ... & 1 \end{bmatrix}$$

so that the Brownian motion at time t_i becomes $\overline{W}_{t_i} = \sum_{j=0}^{i-1}\sqrt{\frac{1}{\Delta t}\int_{t_j}^{t_{j+1}} \sigma_j^2 ds}\hat{W}_{t_j} = \sum_{j=1}^{i}\sigma_j y_j$. In the case where the volatility is constant we get $\sigma_j = \sigma$ and the stock price re-write

Electronic copy available at: https://ssrn.com/abstract=3307566

$$S_{t_i} = S_0 C_{t_i} e^{-\frac{1}{2}\sigma^2(i\times\Delta t)+\overline{W}_{t_i}} = S_0 C_{t_i} \prod_{j=1}^{i} e^{-\frac{1}{2}\sigma^2\Delta t+\sigma\sqrt{\Delta t}y_j}$$

In the case where the volatility is piecewise constant, given $\hat{\sigma}_i$ in the interval $[t_{i-1}, t_i[$ we get

$$\sigma_i^2 = \int_{t_{i-1}}^{t_i} \hat{\sigma}_i^2 ds = \hat{\sigma}_i^2(t_i - t_{i-1})$$

which is the approximation made in the Euler scheme.

15.2 The Monte Carlo engine

Each replication in a Monte Carlo simulation can be interpreted as the result of applying a series of transformations to an input sequence of uniformly distributed random variables U_i with $i = 1, .., d$ producing the output $f(U_1, .., U_d)$. In control theory, f is the result of a transformation converting the U_i to normal random variables, which are transformed to paths of underlying assets, which are transformed to the discounted payoff of the objective. Given the objective value in Equation (15.1.2), the goal is to compute

$$E[f(U_1, .., U_d)] = \int_{[0,1)^d} f(x)dx$$

which is approximated by

$$\int_{[0,1)^d} f(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i) \tag{15.2.5}$$

where n is the number of simulation.

15.2.1 The setup of the Monte Carlo engine

Given a continuous process $(X_t)_{t\geqslant 0}$ defined in Appendix (15.1), we want to estimate

$$\theta = E[f(X_T)]$$

for a fixed maturity T. We take a discrete process $\hat{X}_t$ given by $\{\hat{X}_h, \hat{X}_{2h}, ..., \hat{X}_{mh}\}$ such that $mh = T$. To simplify notation we define $f_j = f(\hat{X})$ and consider the estimated value of

$$\hat{\theta} = E[f(\hat{X}_T)]$$

to be

$$\hat{\theta}_n = \frac{1}{n}\sum_{j=1}^{n} f_j$$

From the strong law of large numbers we get

$$\hat{\theta}_n \to \theta \text{ as } n \to \infty$$

Also, if we consider $Z \sim N(0,1)$ and let $Z_{1-\frac{\alpha}{2}}$ be the $(1-\frac{\alpha}{2})$ percentile point of the $N(0,1)$ distribution so that $P(-Z_{1-\frac{\alpha}{2}} \leqslant Z \leqslant Z_{1-\frac{\alpha}{2}}) = 1-\alpha$ we can recover a confidence interval. The approximate $100(1-\frac{\alpha}{2})\%$ confidence interval for θ when n is large is

Electronic copy available at: https://ssrn.com/abstract=3307566

$$[L(Y), U(Y)] = [\hat{\theta}_n - Z_{1-\frac{\alpha}{2}} \frac{\hat{\sigma}_n}{\sqrt{n}}, \hat{\theta}_n + Z_{1-\frac{\alpha}{2}} \frac{\hat{\sigma}_n}{\sqrt{n}}]$$

where $\hat{\sigma}_n$ is an estimated standard deviation of the process $\hat{X}_t$.

***Remark* 15.2.1** *$\hat{\theta}_n$ is an unbiased estimate of $\hat{\theta}$ but it can be a very biased estimate of θ.*

In order to measure that biase we consider the discretisation error D given by

$$D = |E[f(X_T)] - E[f(\hat{X}_T)]|$$

It leads to two types of error, the discretisation error and the statistical error.

- small value of $m \rightarrow$ greater discretisation error
- small value of $n \rightarrow$ greater statistical error

The values m and n must be chosen judiciously to control the convergence of the estimated $\hat{\theta}_n$ to the true value θ at the minimum computational cost. From the confidence interval, we see that the accuracy of the Monte Carlo pricer is governed by the relation

$$\frac{\hat{\sigma}_n}{\sqrt{n}}$$

so that we can

- multiply n by 4 to decrease the confidence interval by a factor of 2
- reduce the variance

We are now going to briefly state the different techniques that can be put in place to reduce the variance.

15.2.2 Variance reduction techniques

The situation of two square integrable r.v. X and $\tilde{X}$ such that $X \neq \tilde{X}$ having the same expectation

$$E[X] = E[\tilde{X}] = m$$

with non-zero variance $Var(X)$ and $Var(\tilde{X})$ can be reformulated by setting

$$Y = X - \tilde{X} \text{ with } E[Y] = 0 \text{ and } Var(Y) > 0$$

If $Var(X - Y) << Var(X)$ one will choose $X - Y$ to implement the Monte Carlo simulation. The aim of variance reduction techniques is to rewrite the quantity that we want to compute as the expectation of a random variable which has a smaller variance. Suppose that we want to calculate $E[X]$, then we try to re-express it as

$$E[X] = E[Y] + C$$

where Y is a random variable with lower variance than that of X and C is a known constant.

Electronic copy available at: https://ssrn.com/abstract=3307566

15.2.2.1 Control variates

The idea of control vatiates is to write $E[f(X)]$ as

$$E[f(X)] = E[f(X) - \nu g(X)] + \nu E[g(X)] = E[\tilde{f}(X)]$$

where $\overline{g} = E[g(X)]$ can be explicitly computed and

$$var(f(X) - \nu g(X)) = var(f(X)) + \nu^2 var(g(X)) - 2\nu Cov(f, g)$$

is smaller than $var(f(X))$. This happens if f and g are strongly correlated. If we set

$$\nu^* = \frac{Cov(f, g)}{var(g)}$$

we get

$$var(\tilde{f}) = \left(1 - \rho_{f,g}^2\right) var(f)$$

which guarantees variance reduction if f and g are correlated.

15.2.2.2 Perfect control variates

In theory, a perfect control variate exist for diffusion models. Assume we want to compute $E[Z]$ where $Z = \psi(X_s, 0 \leqslant s \leqslant T)$ and X_t is the solution of

$$dX_t = b(X_t)dt + \sigma(X_t)dW_t \ , X(0) = x$$

We can use the predictable representation theorem to find means of decreasing the variance of the Monte Carlo method practically.

Theorem 15.2.1 *Let Z be a random variable measurable with respect to the σ-field generated by the Brownian motion and such that $E[Z^2] < \infty$. Then there exists a stochastic process $(H_t)_{t \leqslant T}$ adapted to the Brownian filtration such that $E[\int_0^T H_s^2 ds] < \infty$ and*

$$Z = E[Z] + \int_0^T H_s dW_s$$

Eventhough we can cancel the variance of Z, the explicit computation of $(H_s)_{s \leqslant T}$ is much more complicated than $E[Z]$. In the special case X_t is a Markov process, under some circonstances the process (X_t) can be written as $H_t = v(t, X_t)$ a function of t and x. If we let b and σ be two Lipschitz functions, and A its infinitesimal generator of the diffusion

$$Af(x) = \frac{1}{2} \sum_{i,j=1}^{n} a_{ij}(x) \partial_{x_i x_j}^2 f(x) + \sum_{j=1}^{n} b_j(x) \partial_{x_j} f(x)$$

where $a_{ij}(x) = \sum_{k=1}^{p} \sigma_{ik}(x)\sigma_{jk}(x)$. Assume that there exists a $C^{1,2}([0, T] \times R^d)$ function with bounded derivatives in x solution to

$$\begin{aligned} &\partial_t u(t, x) + Au(t, x) = f(x) \\ &u(T, x) = g(x) \end{aligned}$$

Then, if $Z = g(X_T) - \int_0^T f(X_s)ds$ and

Electronic copy available at: https://ssrn.com/abstract=3307566

$$Y = \int_0^T \partial_x u(s, X_s)\sigma(s, X_s)dW_s$$

we have

$$E[Z] = Z - Y$$

and Y is a perfect control variate of Z. Hence, we only need to look for H_t as a function of t and X_t. In a practical situation, we assume that we want to compute $u(0, x) = E[Z]$ with $Z = g(X_T) - \int_0^T f(X_s)ds$ and that we know a rough approximation $\overline{u}$ for u. We can therefore choose

$$Y = \int_0^T \partial_x \overline{u}(s, X_s)\sigma(s, X_s)dW_s \tag{15.2.6}$$

as a control variate. For every choice of $\overline{u}$ we obtain an unbiased estimator for $E[Z]$ by setting $\tilde{Z} = Z - Y$. For a reasonable choice of $\overline{u}$ we can expect an improvement of the variance of the estimator. If we set

$$\overline{Z} = g(X_T) - \int_0^T \partial_x \overline{u}(s, X_s)\sigma(s, X_s)dW_s$$

then $\overline{Z}$ is an unbiased estimator of $E[g(X_T)]$ and

$$E[|\overline{Z} - E[g(X_T)]|^2] = E\Big[\int_0^T |\partial_x u(t, X_s) - \partial\overline{u}(t, X_s)|^2\sigma^2(s, X_s)ds\Big]$$

The variance is small if $\partial_x\overline{u}()$ is a good approximation of $\partial_x u()$. Hence, one approximates

$$\int_0^T \partial_x u(s, X_s)\sigma(s, X_s)dW_s$$

by the sum

$$\sum_{k=1}^n \partial_x \overline{u}(kh, X_{kh})\sigma(kh, X_{kh})(W_{(k+1)h} - W_{kh})$$

where $h = \frac{T}{n}$.

15.2.2.3 Using the hedge as control variates

The idea is to use an approximate gradient of the objective function as a control variate. For example, we let $(S_t)_{t\geqslant 0}$ be the state of the system with dynamics being

$$\frac{dS_t}{S_t} = rdt + \sigma(t, S_t)dW_t$$

and we let $C(t, S_t)$ be the objective value on that state. We assume that an explicit solution of that objective is known and given by $\overline{C}(t, S_t)$. Considering the control variate in Equation (15.2.6), it becomes

$$Y = \int_0^T \partial_x \overline{C}(s, S_s)\sigma(s, S_s)S_s dW_s = \int_0^T \partial_x \overline{C}(s, S_s)\big(dS_s - E[dS_s]\big)$$

Then we can use the random variable

$$Y = \sum_{k=1}^n \frac{\partial\overline{C}}{\partial x}(t_k, S_{t_k})\big((S_{t_{k+1}} - S_{t_k}) - E[S_{t_{k+1}} - S_{t_k}|\mathcal{F}_{t_k}]\big)$$

Electronic copy available at: https://ssrn.com/abstract=3307566

as control variate, which simplifies to

$$Y = \sum_{k=1}^{n} \frac{\partial \overline{C}}{\partial x}(t_k, S_{t_k})\big(S_{t_{k+1}} - E[S_{t_{k+1}}|\mathcal{F}_{t_k}]\big)$$

If $\overline{C}$ is close to C and if n is large enough, we can obtain a very large reduction of the variance. Given the solution

$$S_{t_{k+1}} = S_{t_k} e^{\int_{t_k}^{t_{k+1}} r_s ds} e^{-\frac{1}{2}\int_{t_k}^{t_{k+1}} \sigma^2(s,S_s)ds + \int_{t_k}^{t_{k+1}} \sigma(s,S_s)dW_s}$$

we get $E[S_{t_{k+1}}|S_{t_k}] = S_{t_k} e^{\int_{t_k}^{t_{k+1}} r_s ds}$.

15.2.2.4 Importance sampling

Here, we change the sampling law. Suppose we want to calculate $\mu = E_{x\sim f(x)}[g(X)]$ where X is a random variable with density $f(x)$ on $\mathbb{R}$. Then the expectation rewrite as

$$\begin{aligned} E_{x\sim f(x)}[g(X)] &= \int g(x)f(x)dx \\ &= \int g(x)\frac{f(x)}{\tilde{f}(x)}\tilde{f}(x)dx \\ &= E_{x\sim \tilde{f}(x)}[g(X)\frac{f(X)}{\tilde{f}(X)}] \end{aligned}$$

where X has density $\tilde{f}(x) > 0$ under $\mathcal{P}$. By making a multiplicative adjustment to g we compensate for sampling from $\tilde{f}$ instead of f. The adjustment factor $\frac{f(x)}{\tilde{f}(x)}$ is called the likelihood ratio. The distribution $\tilde{f}$ is the importance distribution, and f is the nominal distribution. The importance sampling estimate of μ is

$$\widehat{\mu}_{\tilde{f}} = \frac{1}{n}\sum_{i=1}^{n} g(X_i)\frac{f(X_i)}{\tilde{f}(X_i)}\,, X_i \sim \tilde{f}$$

Theorem 15.2.2 *Let $\widehat{\mu}_{\tilde{f}}$ be given above where $\mu = E_{x\sim f(x)}[g(X)]$ and $\tilde{f}(x) > 0$ whenever $g(x)f(x) \neq 0$. Then $E_{x\sim \tilde{f}(x)}[\widehat{\mu}_{\tilde{f}}] = \mu$ and $var_{\tilde{f}}(\widehat{\mu}_{\tilde{f}}) = \frac{\sigma^2_{\tilde{f}}}{n}$ where*

$$\begin{aligned} \sigma^2_{\tilde{f}} &= \int \frac{\big(g(x)f(x)\big)^2}{\tilde{f}(x)}dx - \mu^2 \\ &= \int \frac{\big(g(x)f(x) - \mu\tilde{f}(x)\big)^2}{\tilde{f}(x)}dx \end{aligned}$$

The variance is small when the numerator is close to zero, that is, when $\tilde{f}(x) \propto g(x)f(x)$. Thus, setting $\tilde{f}(x) = \frac{g(x)f(x)}{\mu}$, then $\sigma_{\tilde{f}} = 0$.
Denoting $Z = g(X)\frac{f(X)}{\tilde{f}(X)}$, then if

$$\tilde{f}(x) = \frac{g(x)f(x)}{E_{x\sim f(x)}[g(X)]}$$

we get $var(Z) = 0$.
To form a confidence interval for μ we need to estimate $\sigma^2_{\tilde{f}}$. From the second formula of the above theorem we have

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\sigma_{\tilde{f}}^2 = E_{x\sim\tilde{f}}[\frac{\big(g(X)f(X) - \mu\tilde{f}(X)\big)^2}{\tilde{f}^2(x)}]$$

Since x_i are sampled from $\tilde{f}$ the natural variance estimate is

$$\begin{aligned}\widehat{\sigma}_{\tilde{f}}^2 &= \frac{1}{n}\sum_{i=1}^{n}\big(\frac{g(x_i)f(x_i)}{\tilde{f}(x_i)} - \widehat{\mu}_{\tilde{f}}\big)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\big(w_i g(x_i) - \widehat{\mu}_{\tilde{f}}\big)^2\end{aligned}$$

where $w_i = \frac{f(x_i)}{\tilde{f}(x_i)}$. As an example, a 99% confidence interval for μ is $\widehat{\mu}_{\tilde{f}} \pm 2.58\frac{\sigma_{\tilde{f}}}{\sqrt{n}}$.

15.2.2.5 Antithetic variables

This technique is often efficient but its gains are less dramatic than other variance reduction techniques. If X is a random variable taking values in $\mathbb{R}^d$ and T is a transformation of $\mathbb{R}^d$ such that the law of $T(X)$ is the same as the law of X, we can construct an antithetic method using the equality

$$E[g(X)] = \frac{1}{2}E[g(X) + g(T(X))]$$

15.2.3 The speed of convergence

15.2.3.1 The Richardson extrapolation

The two-point Richardson extrapolation formula in the case of linear convergence is

$$\begin{aligned}P &= aP_k + bP_n \\ a &= \frac{k}{k-n} \text{ and } b = 1 - a\end{aligned}$$

where P_n is the price with n steps and p_k is the price with $k > n$ steps. In the special case where $k = 2n$ the extrapolation simplifies to

$$P = 2P_{2n} - P_n$$

In the case where we have square-root convergence, the two-point Richardson extrapolation formula becomes

$$\begin{aligned}P &= aP_k + bP_n \\ a &= \frac{\sqrt{k}}{\sqrt{k}-\sqrt{n}} \text{ and } b = 1 - a\end{aligned}$$

and in the special case where $k = 2n$ the extrapolation simplifies to

$$P = 3.414P_{2n} - 2.414P_n$$

Electronic copy available at: https://ssrn.com/abstract=3307566

15.2.3.2 The Romberg methods for Euler and Milshtein scheme

A far better way to implement the convergence speed of a given scheme is the use of Romberg extrapolation method. It provides an approach to take optimally advantage of the weak rate of convergence, including in their higher oder form. To simplify, if the coordinates of b and σ are functions of class C^4 with bounded derivatives up to order 4, then there exist constants C_1 and C_2 independent of $h = \frac{T}{m}$ such that the Euler and Milshtein schemes satisfy

$$|E[f(X_T)] - E[f(\hat{X}_{mh})]| \leqslant C_1 e^{C_2 T} h + \wr(h^2)$$

Talay et al proved that the weak discretisation error of the Euler and Milshtein schemes can be expanded in terms of $\frac{1}{m}$. We can summarise their results as follow. For all time steps $h = \frac{T}{m}$, the error at time T corresponding to the Euler scheme can be expanded as

$$E[f(X_T)] - E[f(\hat{X}_{mh})] = C(f)h + \wr(h^2)$$

where $C(f)$ is a constant which can be computed as a function of f. This expansion allow us to apply the Romberg-Richardson extrapolation technique. Put another way, we have

$$|E[f(X_T)] - (2E[f(\hat{X}_{mh})] - E[f(\hat{X}_{m2h})])| \leqslant K_T h^2$$

where in the first case $mh = T$ and in the second case $m2h = T$. The numerical cost of the Romberg-Richardson procedure is much smaller than the cost corresponding to schemes of order $\frac{1}{m^2}$. If f is smooth enough, it can be shown that for any integer $k > 0$ there exists constants $C_1, .., C_{k+1}$ depending on f such that

$$E[f(X_T)] - E[f(\hat{X}_{mh})] = \frac{C_1(f)}{m} + \frac{C_2(f)}{m^2} + ... + \frac{C_k(f)}{m^k} + \frac{R_{mh}}{m^{k+1}}$$

with $|R_{mh}|$ bounded. Going further in the analysis, if b and σ are functions of class C^4 with bounded derivatives up to order 4, and $f : \mathbb{R}^d \to \mathbb{R}$ is 4 times differentiable with polynomial growth, then

$$E[f(X_T^x)] - E[f(\hat{X}_T^x)] = \mathcal{O}(\frac{1}{m}) \text{ as } m \to \infty$$

Assuming more regularity on the coefficients or some uniform ellipticity on the diffusion coefficient σ one can obtain an expansion of the error at any order. Assume b and σ are infinitely differentiable with bounded partial derivatives, and $f : \mathbb{R}^d \to \mathbb{R}$ is infinitely differentiable with partial derivatives having polynomial growth. Then, given $h = \frac{T}{n}$ for every $R \geqslant 1$

$$\epsilon_{R+1} = E[f(\hat{X}_T)] - E[f(X_T)] = \sum_{k=1}^{R} \frac{c_k}{n^k} + \mathcal{O}(n^{-(R+1)}) \text{ as } n \to \infty$$

where the real coefficients c_k depend on f, T, b and σ. We let $f \in V$ where V is a vector space of continuous functions with linear growth. Given $X^1 = X$, we consider a regular MC simulation based on M copies $(\hat{X}_T)^m$ for $m = 1, .., M$ of the Euler scheme with step $h = \frac{T}{n}$ induces the global squared quadratic error

$$|E[f(X_T)] - \frac{1}{M} \sum_{m=1}^{M} f((\hat{X}_T)^m)|_2^2 = \frac{c_1^2}{n^2} + \frac{Var(f(\hat{X}_T))}{M} + \mathcal{O}(n^{-3})$$

We consider a second Brownian Euler scheme X^2 written with respect to a second Brownian motion W^2 defined on the same probability space $(\Omega, \mathcal{A}, \mathbb{P})$. We assume it has a twice smaller step $\frac{T}{2n}$ and is denoted $(\hat{X})^2$, then assuming (ϵ_2^V) to be more precise

$$E[f(X_T)] = E[2f((\hat{X}_T)^2) - f((\hat{X}_T)^1)] - \frac{1}{2}\frac{c_2}{n^2}\mathcal{O}(n^{-3})$$

Electronic copy available at: https://ssrn.com/abstract=3307566

we get the new global squared quadratic error

$$|E[f(X_T)] - \frac{1}{M}\sum_{m=1}^{M} 2f(((\hat{X}_T)^2)^m) - f(((\hat{X}_T)^1)^m)|_2^2 = \frac{c_2^2}{4n^4} + \frac{Var(2f((\hat{X}_T)^2) - f((\hat{X}_T)^1))}{M} + \mathcal{O}(n^{-5})$$

We need to find a way to control the term $Var(2f((\hat{X}_T)^2) - f((\hat{X}_T)^1))$. It is shown that if $W^2 = W^1 = W$, then

$$Var(2f(\hat{X}_T^2) - f(\hat{X}_T^1)) \to Var(2f(X_T) - f(X_T)) = Var(f(X_T)) \text{ as } n \to \infty$$

and that this choice is optimal among all possible choice of correlated Brownian motion W^1 and W^2. This result can be extended to Borel functions f when the diffusion is uniformly elliptic. Hence, one first simulates an Euler scheme with step $\frac{T}{2n}$ using a white Gaussian noise $(U_k^2)_{k\geqslant 1}$, then one simulates a Euler scheme with step $\frac{T}{n}$ using the white Gaussian noise

$$U_k^1 = \frac{U_{2k}^2 + U_{2k-1}^2}{\sqrt{2}}, k \geqslant 1$$

If one adopts the lazy approach based on independent Gaussian noises U^1 and U^2, then the asymptotic variance is

$$Var(2f(X_T^2) - f(X_T^1)) = 4Var(f(X_T^2)) + Var(f(X_T^1)) = 5Var(f(X_T^1))$$

A multistep Romberg extrapolation with consistent Brownian increments was developed. In the case $R = 3$, we set

$$\alpha_1 = \frac{1}{2}, \alpha_2 = -4, \alpha_3 = \frac{9}{2}$$

such that

$$E\big[\alpha_1 f(\hat{X}_T^1) + \alpha_2 f(\hat{X}_T^2) + \alpha_3 f(\hat{X}_T^3)\big] = \frac{c_3^3}{n^3} + \mathcal{O}(n^{-4})$$

where $\hat{X}^r$ denotes the Euler scheme with step $\frac{T}{rn}$ for $r = 1, 2, 3$ with respect to the same Brownian motion W. Eventhough this choice induces a control of the variance of the estimator, it is no-longer optimal yet it is natural. The most efficient way to simulate the three white noises $(U_k^r)_{1\leqslant k\leqslant r}$ on one time step $\frac{T}{n}$ is to let U_i for $i = 1, 2, 3, 4$ be four i.i.d. copies of $N(0, I_q)$ and set

$$\begin{aligned} U_1^3 &= U_1, U_2^3 = \frac{U_2 + U_3}{\sqrt{2}}, U_3^3 = U_4 \\ U_1^2 &= \frac{\sqrt{2}U_1 + U_2}{\sqrt{3}}, U_2^2 = \frac{U_3 + \sqrt{2}U_4}{\sqrt{3}} \\ U_1^1 &= \frac{U_1^2 + U_2^2}{\sqrt{2}} \end{aligned}$$

15.2.4 Weak error for path-dependent functionals

We consider some path-dependent European options related to some payoffs $F((X_t)_{t\in[0,T]})$ where F is a functional defined on the set $\mathbb{D}([0,T],\mathbb{R}^d)$ of right continuous left-limited functions $x : [0,T] \to \mathbb{R}$. All the asymptotic control of the variance obtained in the previous section for the estimator $\sum_{r=1}^R \alpha_r f(\hat{X}_T^r)$ of $E[f(X_T)]$ when f is continuous can be extended to functionals $F : \mathbb{D}([0,T],\mathbb{R}^d) \to \mathbb{R}$ which are $\mathbb{P}_X$-a.s.continuous with respect to the sup-norm defined by $|x|_{sup} = \sup_{t\in[0,T]} |x(t)|$. This is because the continuous Euler scheme $\hat{X}$ with step $\frac{T}{n}$ defined by

Electronic copy available at: https://ssrn.com/abstract=3307566

$$\hat{X}_t = x_0 + \int_0^t b(\hat{X}_{\overline{s}})ds + \int_0^t \sigma(\hat{X}_{\overline{s}})dW_s \ , \ \overline{s} = \frac{ns}{T}$$

converges for the sup-norm toward X in every $L^P(\mathbb{P})$. Furthermore, the asymptotic control of the variance holds true with any R-tuple $\alpha = (\alpha_r)_{1 \leqslant r \leqslant R}$ of weights coefficients satisfying $\sum_{1 \leqslant r \leqslant R} \alpha_r = 1$ so that these coefficients can be adapted to the structure of the weak error expansion.

On the other hand, several papers provided some weak rates of convergence for some families of functionals F. For instance, in the case of barrier options

$$F(x) = \Phi(x(T))I_{\tau_D(x) \leqslant T}$$

where Φ is at least Lipschitz and $\tau_D = \inf\{s \in [0,T], x(s) \in D^c\}$ is the hitting time of D^c by x. For instance, if the domain D has smooth enough boundary, $b, \sigma \in C^3(\mathbb{R}^d)$, σ uniformly elliptic on D, then for every Borel bounded function f vanishing in a neighbourhood of ∂D then

$$E[f(\hat{X}_T)I_{\tau(\hat{X}) > T}] - E[f(X_T)I_{\tau(X) > T}] = \mathcal{O}(\frac{1}{\sqrt{n}}) \text{ as } n \to \infty$$

If further, b and σ are C^5 then

$$E[f(\hat{X}_T)I_{\tau(\hat{X}) > T}] - E[f(X_T)I_{\tau(X) > T}] = \mathcal{O}(\frac{1}{n}) \text{ as } n \to \infty$$

but these assumptions are not satisfied by usual barrier options. A similar improvement to $\mathcal{O}(\frac{1}{n})$ rate can be expected when replacing $\hat{X}$ by the continuous Euler scheme $\overline{X}$. For both classes of functionals, the practical implementation of the continuous Euler scheme is known as the Brownian Bridge method.

15.2.5 Romberg extrapolation for path-dependent functionals

There are two ways to implement the (multistep) Romberg extrapolation with consistent Brownian increments in order to improve the performance of the original (stepwise constant or continuous) Euler schemes. Both rely on natural conjectures about the existence of a higher order expansion of the time discretisation error suggested by the above rates of convergence.

15.2.5.1 The stepwise constant Euler scheme

For the stepwise constant Euler scheme, it means the existence of a vector space V (stable by product) of admissible functionals satisfying for all $F \in V$

$$(\epsilon_R^{\frac{1}{2},V}) \to E[F(X)] = E[F(\hat{X})] + \sum_{k=1}^{R-1} \frac{c_k}{n^{\frac{k}{2}}} + \mathcal{O}(n^{-\frac{R}{2}})$$

For small values of R, one gets

$$R = 2 \ \alpha_1^{\frac{1}{2}} = -(1+\sqrt{2}) \ , \ \alpha_2^{\frac{1}{2}} = \sqrt{2}(1+\sqrt{2})$$
$$R = 3 \ \alpha_1^{\frac{1}{2}} = \frac{\sqrt{3}-\sqrt{2}}{2\sqrt{2}-\sqrt{3}-1} \ , \ \alpha_2^{\frac{1}{2}} = -2\frac{\sqrt{3}-1}{2\sqrt{2}-\sqrt{3}-1} \ , \ \alpha_3^{\frac{1}{2}} = 3\frac{\sqrt{2}-1}{2\sqrt{2}-\sqrt{3}-1}$$

However, these coefficients have greater absolute values than in the standard case. Thus if $R = 4$, then $\sum_{1 \leqslant r \leqslant 4} (\alpha_r^{\frac{1}{2}})^2 \approx 10900!$ which induces an increase of the variance term for too small values of the time discretisation parameters n even when increments are consistently generated.

Electronic copy available at: https://ssrn.com/abstract=3307566

15.2.5.2 The continuous Euler scheme

The conjecture is simply to assume that the expansion (ϵ_R) now holds for a vector space V of functionals F with polynomial growth with respect to the sup-norm. But the increase of the complexity induced by the Brownian Bridge method is difficult to quantise. It amounts to computing $\log U_k$ and the inverse distribution $F^{-1}_{x,y}$ and $G^{-1}_{x,y}$. Moreover, simulating the extrema of some continuous Euler schemes using the Brownian Bridge in a consistent way is not straightforward. But, one can reasonably expect that using independent Brownian Bridges relying on stepwise constant Euler schemes with consistent Brownian increments will have a small impact on the global variance (although slightly increasing it). So simulate the stepwise constant Euler scheme $\hat{X}^1$ and $\hat{X}^2$ and use consistent Brownian increments. Between two points t^n_k and t^n_{k+1} apply the BB to find the probability of crossing the barrier. Repeat the approach between the points t^{2n}_k and t^{2n}_{k+1}.

When the volatility is high, and $M = 10^6$ then the quasi-consistent 3-step Romberg extrapolation with Brownian Bridge clearly outperform the continuous Euler scheme (Brownian Bridge) of equivalent complexity while the 3-step Romberg extrapolation based on the stepwise constant Euler schemes with consistent Brownian increments is not competitive at all. It suffers from both a too high variance for the considered sizes of the MC simulation and from its too slow rate of convergence in time.

Electronic copy available at: https://ssrn.com/abstract=3307566

Bibliography

[1989] Aarst E.H.L., Korst J., Simulated annealing and Boltzmann machines. John Wiley.

[2012] Abernethy J., Chen Y., Vaughan J.W., Efficient market making via convex optimization, and a connection to online learning. Working Paper, University of California, Berkeley.

[1993] Abry P., Goncalves P., Flandrin P., Wavelet-based spectral analysis of $1/f$ processes. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, **3**, pp 237–240.

[1995] Abry P., Goncalves P., Flandrin P., Wavelets, spectrum estimation and $1/f$ processes. in A. Antoniadis and G. Oppenheim, eds, *Lecture Notes in Statistics: Wavelets and Statistics*, Springer-Verlag, pp 15–30.

[1996] Abry P., Sellan F., The wavelet-based synthesis for fractional Brownian motion. proposed by F. Sellan and Y. Meyer, Remarks and Fast Implementation, *Applied and Computational Harmonic Analysis*, **3**, (4), pp 377–383.

[1998] Abry P., Veitch D., Wavelet analysis of long-range-dependent traffic. *IEEE Transaction on Information Theory*, **44**, (1), pp 2–15.

[1999] Abry P., Sellan F., A wavelet-based joint estimator of the parameters of long-range dependence. *IEEE Transaction on Information Theory*, **45**, (3), pp 878–897.

[2000] Abry P., Flandrin P., Taqqu M.S., Veitch D., Wavelets for the analysis, estimation, and synthesis of scaling data. in *Self-similar Network Traffic and Performance Evaluation*, ed. K. Park, W. Willinger, Wiley, pp 39–87.

[2002] Abry P., Baraniuk R., Flandrin P., Riedi R., Veitch D., Multiscale nature of network traffic. *IEEE Signal Processing Magazine*, **19**, (3), pp 28–46.

[2001] Abu-Mostafa Y.S., Atiya A.F., Magdon-Ismail M., White H., Neural networks in financial engineering. *IEEE Transactions on Neural Networks*, **12**, (4), pp 653–656.

[1998] Adya M., Collopy F., How effective are neural networks at forecasting and prediction? A review and evaluation. *Journal of Forecasting*, **17**, pp 481–495.

[2010] Ahmed N.K., Atiya A.F., El Gayar N., El-Shishiny H., An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, **29**, (5), pp 594–621.

[2010] Alpaydin E., Introduction to machine learning. MIT Press

[1997] Amit Y., Geman D., Shape quantization and recognition with randomized trees. *Neural Computation*, **9**, (7), pp 1545–1588.

[1996] Anders U., Korn O., Schmitt C., Improving the pricing of options: a neural network approach. ZEW Discussion Papers, pp 96–104.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2015] Antonik P., Duport F., Smerieri A., Haelterman P., Massar S., FPGA implementation of reservoir computing with online learning. Université Libre de Bruxelles.

[2016] Antonik P., Hermans M., Duport F., Haelterman M., Massar S., Towards pattern generation and chaotic series prediction with photonic reservoir computers. Laboratoire d'Information Quantique, Universite Libre de Bruxelles, Belgium.

[1974] Appel G., Double your money every three years. Brightwaters, NY: Windsor Books.

[1999] Appel G., Technical analysis power tools for active investors. Financial Times Prentice Hall.

[2008] Appel G., Appel M., A quick tutorial in MACD: Basic concepts. Working Paper.

[2006] Applegate D.L., Bixby R.E., Chvatal V., Cook W.J., Concorde tsp solver. Working Paper.

[2011] Applegate D.L., Bixby R.E., Chvatal V., Cook W.J., The traveling salesman problem: A computational study. Princeton university press, 2011.

[2001] Armstrong J.S., Principles of forecasting: A handbook for researchers and practitioners. ed.: Norwell, MA: Kluwer Academic Publishers.

[1991] Arneodo A., Bacry E., Muzy J-F., Wavelets and multifractal formalism for singular signals: Application to turbulence data. *Phys. Rev. Lett.*, **67**, pp 3515–3518.

[1995] Arneodo A., Bacry E., Graves P.V., Muzy J-F., Characterizing long-range correlations in DNA sequences from wavelet analysis. *Phys. Rev. Lett.*, **74**, 3293.

[1998] Arneodo A., Muzy J-F., Dornette D., Discrete causal cascade in the stock market. *Eur. Phys. J.*, **2**, pp 277–282.

[2017] Arras L., Montavon G., Muller K-R., Samek W., Explaining recurrent neural network predictions in sentiment analysis. In *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pp. 159–168.

[2000] Atiya A.F., Parlos A.G., New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on neural networks*, **11**, (3), pp 697–709.

[1996] Aytug H., Koehler G.J., New stopping criterion for genetic algorithms. Working Paper, University City Blvd and University of Florida.

[2001] Bacry E., Delour J., Muzy J-F., Multifractal random walk. *Physical Review E*, **64**, 026103–026106.

[2015] Bahdanau D., Cho K., Bengio Y., Neural machine translation by jointly learning to align and translate. In ICLR, 2015.

[2006] Bai R., Wang X., An effective hybrid classifier based on rough set and neural network. *International Conference IEEE*, 0-7695-2749-3/06.

[1996] Baillie R.T., Long memory processes and fractional integration in econometrics. *Journal of Econometrics*, **73**, pp 5–59.

[1996] Baillie R.T., Bollerslev T., Mikkelsen H.O., Fractionally integrated generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, **74**, pp 3–30.

[2007] Balamurugan R., Subramanian S., Self-adaptive differential evolution based power economic dispatch of generators with valve-point effects and multiple fuel options. *International Journal of Computer Science and Engineering*, Winter.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2009] Bandurski K., Kwedlo W., Training neural networks with hybrid differential evolution algorithm. Zeszyty Naukowe Politechniki Bialostockiej.

[1988] Barron A.R., Barron R.L., Statistical learning networks: A unifying view. in E. Wegman, D. Gantz, and J. Miller, eds. *20th Symposium on the Interface: Computing Science and Statistics*, (American Statistical Association, Reston, Virginia) pp 192–203.

[2012] Barunik J., Understanding the source of multifractality in financial markets. *Physica A*, **391**, (17), pp 4234–4251.

[2013] Battula B.P., Satya Prasad R., An overview of recent machine learning strategies in data mining. *International Journal of Advanced Computer Science and Applications*, **4**, (3), pp 50–54.

[1993] Bazaraa M.S., Sherali H.D., Shetty C.M., Nonlinear programming theory and applications. 2nd ed. John Wiley and Sons.

[2008] Bekiros S.D., Georgoutsos D.A., Direction-of-change forecasting using a volatility based recurrent neural network. *Journal of Forecasting*, **27**, (5), pp 407–417.

[1999] Bellgard C., Goldschmidt P., Forecasting foreign exchange rates: Random walk hypothesis, linearity and data frequency. Working Paper, The University of Western Australia.

[2017] Bello I., Pham H., Le Q.V., Norouzi M., Bengio S., Neural combinatorial optimizastion with reinforcement learning. in a conference paper at ICLR 2017.

[1994] Bengio Y., Simard P., Frasconi P., Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, **5**, (2), pp 157–166.

[2001] Benhamou E., Delta equivalent asset flows. Swaps Strategy, Goldman Sachs International, London.

[2004] Bennell J., Sutcliffe C., Black-Scholes versus Arti

cial Neural Networks in Pricing FTSE 100 Options. *Intelligent Systems in Accounting*, Finance and Management, **12**, pp 243–260.

[1996] Bertsekas D.P., Constrained optimization and Lagrange multiplier methods. MIT, Athena Scientific.

[2005] Bertsekas D.P., Dynamic programming and optimal control. volume 1. Third Edition, Athena Scientific.

[2015] Bertsekas D.P., Convex optimization algorithms. Belmont, MA., Athena Scientific.

[1998] Bessembinder H., Chan K., Market efficiency and the returns to technical analysis. *Financial Management*, **27**, (2), pp 5–27.

[2009] Beynon-Davies P., Business information systems. Basingstoke, UK

[2012] Bhandari D., Murthy C.A., Pal S.K., Variance as a stopping criterion for genetic algorithms with elitist model. *Fundamenta Informaticae*, **120**, pp 145–164.

[2015] Bhardwaj A., Tiwari A., Breast cancer using genetically optimized neural network model. *International journal Expert System with Application*.

[2017] Bianchi F.M., Kampffmeyer M., Maiorino E., Jensen R., Temporal overdrive recurrent neural network. Working Paper, University of Tromso, arXiv:1701.05159v1.

[1978] Binder D.A., Bayesian cluster analysis. *Biometrika*, **65**, pp 31–38.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1931] Birkhoff G.D., Proof of the ergodic theorem. *Proc. Natl. Acad. Sci. USA*, **17**, (12), pp 656–660.

[2006] Bishop C.M., Pattern recognition and machine learning. Springer Verlag.

[2001] Blacher G., A new approach for designing and calibrating stochastic volatility models for optimal delta-vega hedging of exotic options. Conference presentation at Global Derivatives and Risk Management, Juan-les-Pins.

[1973] Black F., Scholes M., The pricing of options and corporate liabilities. *Journal of Political Economics*, **81**, pp 637–659.

[2010] Bloch D.A., A practical guide to implied and local volatility Working Paper, University of Paris 6,, SSRN-id1538808.

[2011] Bloch D.A., Coello Coello C.A., Smiling at evolution. *Applied Soft Computing*, **11**, (8), pp 5724–5734.

[2012] Bloch D., From implied to local volatility surface. Working Paper, University of Paris 6 Pierre et Marie Curie.

[2014] Bloch D., A practical guide to quantitative portfolio trading. Working Paper, University of Paris 6 Pierre et Marie Curie.

[1986] Bollerslev T., Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, **31**, pp 307–327.

[1999] Bollerslev T., Jubinski D., Equality trading volume and volatility: Latent information arrivals and common long-run dependencies. *Journal of Business & Economic Statistics*, **17**, pp 9–21.

[1992] Bollinger J., Using Bollinger bands. *Technical Analysis of Stocks and Commodities*, **10**, February.

[2003] Bondarenko O., Estimation of risk-neutral densities using positive convolution approximation. *Journal of Econometrics*, **116**, pp 85–112.

[1992] Boser B.E., Guyon I.M., Vapnik V.N., A training algorithm for optimal margin classifiers. in *Proceedings of the fifth annual workshop on Computational learning theory*, pp 144.

[2017] Bouizi R., Financial time series forecasting using wavelet transform and reservoir computing paradigm. Quant Finance Ltd, MSc Thesis, Department of Mathematics, Imperial College London.

[2004] Boyd S., Vandenberghe L., Convex optimization. Cambridge University Press.

[2000] Brabazon T., A connectivist approach to index modelling in financial markets. Department of Accountancy, University College, Dublin.

[1995] Branicky M.S., Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, **138**, pp 67–100.

[1998] Breidt F.J., Crato N., de Lima P., On the detection and estimation of long memory in stochastic volatility. *Journal of Econometrics*, **83**, pp 325–348.

[1984] Breiman L., Friedman J., Stone C.J., Olshen R.A., Classification and regression trees. Chapman and Hall/CRC, Boca Raton, FL.

[1996] Breiman L., Stacked regressions. *Machine Learning*, **24**, (1), pp 49–64.

[1996b] Breiman L., Out-of-bag estimation. Technical report, Department of Statistics, University of California.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1996c] Breiman L., Bagging predictors. *Machine Learning*, **24**, (2), pp 123–140.

[2001] Breiman L., Random forests. *Machine learning*, **45**, (1), pp 5–32

[2006] Brest J., Greiner S., Boskovic B., Mernik M., Zumer V., Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, **10**, (6), pp 646–657.

[1992] Brock W., Lakonishok J., LeBaron B., Simple technical trading rules and the stochastic properties of stock returns. *Journal of Finance*, **47**, (5), pp 1731–1764.

[1999] Brown C., Technical analysis for the trading professional. 1st Edition, McGraw-Hill.

[1969] Bryson A.E., Ho Y.C., Applied optimal control. U.K. Blaisdell.

[2015] Bush N.T., Hasegawa H., Training artificial neural network using modification of differential evolution algorithm. *International Journal of Machine Learning and Computing*, **5**, (1), pp 1–6.

[2005] Bush K., Tsendjav B., Improving the richness of echo state features using next ascent local search. in *Proceedings of the Artificial Neural Networks in Engineering Conference*, pp 227–232, St. Louis.

[2006] Bush K., Anderson C., Exploiting iso-error pathways in the N, k-plane to improve echo state network performance.

[1994] Cai J., A Markov model of switching-regime ARCH. *Journal of Business*, **12**, pp 309–316.

[2004] Cajueiro D.O., Tabak B.M., The Hurst exponent over time: Testing the assertion that emerging markets are becoming more efficient. *Physica A*, **336**, pp 521–537.

[2007] Cajueiro D.O., Tabak B.M., Long-range dependence and multifractality in the term structure of LIBOR interest rates. *Physica A*, **373**, pp 603–614.

[1995] Caldwell R.B., Performances metrics for neural network-based trading system development. *NeuroVet Journal*, **3**, (2), pp 22–26.

[2001] Calvet L., Fisher A., Forecasting multifractal volatility. *Journal of Econometrics*, **105**, pp 27–58.

[2002] Calvet L., Fisher A., Multifractality in asset returns: Theory and evidence. *The Review of Economics and Statistics*, **84**, (3), pp 381–406.

[2004] Calvet L., Fisher A., Regime-switching and the estimation of multifractal processes. *Journal of Financial Econometrics*, **2**, pp 44–83.

[2006] Calvet L., Fisher A., Thompson S., Volatility comovement: A multi-frequency approach. *Journal of Econometrics*, **31**, pp 179–215.

[2013] Calvet L., Fisher A., Wu L., Staying on top of the curve: A cascade model of term structure dynamics. Working Paper.

[1997] Campbell J.Y., Lo A.W., MacKinlay A.C., The econometrics of financial markets. Princeton University Press, New jersey.

[2008] Cao M., Qiao P., Neural network committee-based sensitivity analysis strategy for geotechnical engineering problems. *Neural Comput Appl*, **17**, pp 509–519

[2004] Carbone A., Castelli G., Stanley H., Time-dependent Hurst exponents in financial time series. *Physica A*, **344**, pp 267–271.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2007] Carbone A., Algorithm to estimate the Hurst exponent of high-dimensional fractals. Working Paper, Physics Department, Politecnico di Torino.

[1996] Casey M., The dynamics of discrete-time computation with application to recurrent neural networks and finite state machine extraction. *Neural Computation*,**8**, pp 1135–1178.

[2001] Castiglione F., Bernaschi M., Market fluctuations: Simulation and forecasting. Working Paper

[2006] Cesa-Bianchi N., Lugosi G., Prediction, learning, and games. Cambridge University Press.

[2009] Chan E., Quantitative trading: How to build your own algorithmic trading business. John Wiley & Sons, volume 430.

[1994] Chande T.S., Kroll S., The new technical trader. John Wiley, New York.

[2004] Chen Y., Yang B., Dong J., Nonlinear system modelling via optimal design of neural trees. *International Journal of Neural Systems*, **14**, (2), pp 125–137.

[2009] Chen C-W., Huang C-S., Lai H-W., The impact of data snooping on the testing of technical analysis: An empirical study of Asian stock markets. *Journal of Asian Economics*, **14**, (5), pp 580–591.

[2016] Chen Y., Hoffman M.W., Colmenarejo S.G., Denil M., Lillicrap T.P., de Freitas N., Learning to learn for global optimization of black box functions. Working Paper.

[2010] Chitra A., Uma S., An ensemble model of multiple classifiers for time series prediction. *International Journal of Computer Theory and Engineering*, **2**, (3), pp 1793–8201

[2014] Cho K., Van Merrienboer B., Gulcehre C., Bahdanau D., Bougares F., Schwenk H., Bengio Y., Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv: 1406.1078.

[1984] Chua L.O., Lin G.N., Non-linear programming without computation. *IEEE Trans. Circuits and Systems*, CAS-31, pp 182–186.

[2015] Chung J., Gulcehre C., Cho K., Bengio Y., Gated feedback recurrent neural networks. CoRR, abs:1502.02367.

[1993] Cichocki A., Unbehauen R., Neural networks for optimization and signal processing. John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore.

[1983] Clarke F.H., Optimization and nonsmooth analysis. John Wiley & Sons, New York.

[2006] Clegg R.G., A practical guide to measuring the Hurst parameter. *International Journal of Simulation: Systems, Science and Technology*, **7**, (2), pp 3–14.

[2015] Clevert D-A., Unterthiner T., Hochreiter S., Fast and accurate deep network learning by exponential linear units (elus). Working Paper.

[2000] Coello Coello C.A., Constraint-handling using an evolutionary multiobjective optimization technique. *Civil Engineering and Environmental Systems*, Vol. **17**, pp 319–346.

[2002] Coello Coello C.A., Mezura-Montes E., Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatic*, Vol. **16**, pp 193–203.

[2003] Coello Coello C.A., Mezura-Montes E., Increasing successful offspring and diversity in differential evolution for engineering design. *Advanced Engineering Informatic*, Vol. **16**, pp 193–203.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1960] Cohen J., A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, **20**, (1), pp 37–46.

[1994] Connor J.T., Martin R.D., Atlas L.E., Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, **5**, (2), pp 240–254.

[2002] Cont R., Tankov P., Calibration of jump-diffusion option-pricing models: A robust non-parametric approach. Ecole Polytechnique, CMAP, September.

[2016] Contras D., Matei O., Translation of the mutation operator from genetic algorithms to evolutionary ontologies. in *International Journal of Advanced Computer Science and Applications (IJACSA)*, **7**, (1).

[2002] Cortez P., Rocha M., Neves J., A lamarckian approach for neural network training. *Neural Processing Letters*, **15**, pp 105–116.

[2013] Cortez P., Embrechts M.J., Using sensitivity analysis and visualization techniques to open black box data mining models. *Inf Sci*, **225**, pp 1–17

[2003] Costa R.L., Vasconcelos G.L., Long-range correlations and nonstationarity in the Brazilian stock market. *Physica A*, **329**, pp 231–248.

[2017] Culkin R., Das S.R., Machine learning in finance: The case of deep learning for option pricing. Working Paper, Santa Clara University.

[2000] Cunningham P., Carney J., Diversity versus quality in classification ensembles based on feature selection. Technical Report TCD-CS-2000-02, Department of Computer Science, Trinity College Dublin.

[1988] Cybenko G., Continuous valued neural networks with two hidden layers are sufficient. Technical Report, Department of Computer Science, Tufts University, Medford, MA.

[1989] Cybenko G., Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, **2**, (4), pp 303–314

[2003] Dablemont S., Van Bellegem S., Verleysen M., Modelling and forecasting financial time series of tick data by functional analysis and neural networks. Universite catholique de Louvain, Machine Learning Group, DICE, Louvain-la-Neuve, Belgium.

[1993] Dacorogna M.M., Muller U.A., Nagler R.J., Olsen R.B., Pictet O.V., A geographical model for the daily and weekly seasonal volatility in the foreign exchange market. *Journal of International Money and Finance*, **12**, (4), pp 413–438.

[1998] Dacorogna M.M., Muller U.A., Olsen R.B., Pictet O.V., Modelling short-term volatility with GARCH and HARCH models. in C. Dunis and B. Zhou, (eds.), *Nonlinear Modelling of High Frequency Financial Time Series*, John Wiley & Sons, pp 161–176.

[2001] Dacorogna M.M., Gencay R., Muller U.A., Olsen R.B., Pictet O.V., An introduction to high frequency finance. Academic Press, San Diego, CA.

[2006] Daglish T., Hull J., Suo W., Volatility surfaces: Theory, rules of thumb and empirical evidence. Working Paper, August.

[2003] Damodaran A., Investment philosophies. John Wiley & Sons, New York.

[1994] Dana R.A., Jeanblanc-Picque M., Marche financiers en temps continu: valorisation et equilibre. Recherche en Gestion, Economica, Paris.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1998] Darbellay G.A., Predictability: An information-theoretic perspective. in *Signal Analysis and Prediction*, edt, A. Prochazka, J. Uhlir, P.W.J. Rayner and N.G. Kingsbury, pp 249–262.

[2000] Darbellay G.A., Slama M., Forecasting the short-term demand for electricity? Do neural networks stand a better chance? *International Journal of Forecasting*, **16**, pp 71–83.

[1882] Darwin C.R., The variation of animals and plants under domestication. Murray, London, second edition.

[2005] Das S., Konar A., Chakraborty U.K., Two improved differential evolution schemes for faster global search. in *Proceedings of the 2005 conference on Genetic and Evolutionary Computing*, (GECCO 2005), pp 991–998.

[1993] Davis M.H.A., Markov models and optimization. Chapman & Hall, London.

[2002] Day T.E., Wang P., Dividends, nonsynchronous prices, and the returns from trading the Dow Jones Industrial Average. *Journal of Empirical Finance*, **9**, (4), pp 431–454.

[2000] Deb K., An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, **186**, (2/4), pp 311-338.

[2014] De Ona J., Garrido C., Extracting the contribution of independent variables in neural network models: A new approach to handle instability. *Neural Computing and Applications*, **25**, pp 859–869.

[1999] Derman E., Regimes of volatility. *Risk*, **7**, (2).

[2008] Derman E., Lecture Notes 1–12.

[1993] Ding Z., Granger C.W.J., Engle R.F., A long memory property of stock market returns and a new model. *Journal of the Empirical Finance*, **1**, pp 83–106.

[2017] Dingli A., Fournier K.S., Financial time series forecasting: A machine learning approach. *Machine Learning and Applications: An International Journal* (MLAIJ), **4**, No.1/2/3, September.

[2000] Dominey P.F., Ramus F., Neural network processing of natural language: I. Sensitivity to serial, temporal and abstract structure of language in the infant. *Language and Cognitive Processes*, **15**, (1), pp 87–127.

[1953] Doob J.L., Stochastic processes. John Wiley & Sons.

[1999] Douglas S.C. Introduction to adaptive filters. University of Utah. Published by CRC Press LLC.

[1992] Doya K., Bifurcations in the learning of recurrent neural networks. in *Proceedings of IEEE International Symposium on Circuits and Systems*, **6**, pp 2777–2780.

[1996] Drucker H., Burges C.J.C., Kaufman L., Smola A.J., Vapnik V.N., Support vector regression machines. in *Advances in Neural Information Processing Systems 9, (NIPS)*, MIT Press, pp 155–161.

[1999] Drummond C., Hearne T., The lessons. A series of 30 multi-media lessons, Drummond and Hearne Publications, Chicago.

[1994] Dupire B., Pricing with a smile. *Risk*, **7**, pp 18–20.

[1987] Durbin R., Willshaw D., An analogue approach to the Travelling Salesman. *Nature*, **326**, (16), pp 689–691.

[2003] Durrleman V., A note on initial volatility surface. Working Paper.

[1953] Eccles J.G., The Neurophysiological basis of mind. Clarendon, Oxford.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1993] Efron B., Tibshirani R., An introduction to the bootstrap. Chapman & Hall, New York.

[1995] El Hihi S., Bengio Y., Hierarchical recurrent neural networks for long-term dependencies. In *NIPS*, **400**, pp 409.

[1982] Engle R.F., Autoregressive conditional heteroskedasticity with estimates of the variance of U.K. inflation. *Econometrica*, **50**, pp 987–1008.

[1986] Engle R.F., Granger C.W.j., Rice j., Weiss A., Semiparametric estimates of the relation between weather and electricity sales. *Journal of the American Statistical Association*, **81**, pp 310–320.

[1987] Engle R.F., Granger C.W.j., Co-integration and error correction: Representation, estimation, and testing. *Econometrica*, **55**, 2, pp 251–276.

[2005] Ernst D., Geurts P., Wehenkel L., Tree-based batch model reinforcement learning. *Journal of Machine Learning Research*, **6**, pp 405–556.

[2003] Fan H.Y., Lampinen J., A trigonometric mutation operation to differential evolution. *International Journal of Global Optimization*, **27**, pp 105–129.

[1954] Farley B.G, Clark W.A., Simulation of self-organizing systems by digital computer. *IRE Transactions on Information Theory*, **4**, (4), pp 76–84.

[2005] Fengler M.R., Semiparametric modeling of implied volatility. Springer Finance, Springer-Verlag Berlin Heidelberg.

[2006] Feoktistov V., Differential evolution, in search of solutions. Springer, Optimisation and its Applications, **5**.

[2008] Fernandez-Blanco P., Technical market indicators optimization using evolutionary algorithms. In proceedings of the 2008 GECCO conference companion on genetic and evolutionary computation. Atlanta, USA.

[1936] Fisher R.A., The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, **7**, pp 179–188.

[1981] Fleiss J.L., Statistical methods for rates and proportions. John Wiley & Sons, New York, NY, 2nd edition.

[1975] Fleming W., Rishel R., Deterministic and stochastic control. Springer-Verlag, Berlin.

[1966] Fogel L.J., Artificial intelligence through simulated evolution. John Wiley, New York.

[2008] Fok W.W.T., Tam V.W.L., Computational neural network for global stock indexes prediction. *Proceedings of the World Congress on Engineering*, **2**, pp 1171–1175.

[1989] Follmer H., Schweitzer M., Hedging by sequential regression: An introduction to the mathematics of option trading. *ASTIN Bulletin*, **18**, pp 147–160.

[1990] Follmer H., Schweizer M., Hedging of contingent claims under incomplete information. In M.H.A. Davis and R.J. Elliott, editors, *Applied Stochastic Analysis*, pp 389–414, Gordon and Breach, London.

[1988] Fort J.C., Solving a combinatorial problem via self-organizing process: An application of the Kohonen algorithm to the traveling salesman problem. *Biological Cybernetics*, **59**, (1), pp 33–40.

[1986] Fox R., Taqqu M.S., Large-sample properties of parameter estimates for strongly dependent stationary Gaussian time series. *The Annals of Statistics*, **14**, (2), pp 517–532.

[1997] Freund Y., Schapire R.E., A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, **55**, (1), pp 119–139.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1998] Friedman J.H., Data mining and statistics: What's the connection? *Computing Science and Statistics*, **29**, (1), pp 3–9.

[2000] Friedman J., Hastie T., Tibshirani R., Additive logistic regression: A statistical view of boosting (with discussions). *Annals of Statistics*, **28**, (2), pp 337–407.

[2005] Fu M.C., Glover F.W., April J., Simulation optimization: A review, new developments, and applications. In *Proceedings of the 37th conference on Winter simulation*, Winter Simulation Conference pp 83–95.

[2010] Fua P., Aydin V., Urtasun R., Salzmann M., Least-squares minimization under constraints. Technical Report EPFL-REPORT-150790.

[1980] Funahashi K., Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, **36**, pp 193–202.

[1989] Funahashi K., On the approximate realization of continuous mappings by neural networks. *Neural Network*, **2**, (3), pp 183–192.

[1993] Funahashi K., Nakamura Y., Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, **6**, pp 801–806.

[1993] Gee A.H., Problem solving with optimization networks. PhD thesis.

[1992] Geman S., Bienenstock E., Doursat R., Neural networks and the bias/variance dilemma. *Neural Computation*, **4**, pp 1–58.

[1997] Gencay R., The predictability of security returns with simple technical trading rules. *Journal of Empirical Finance*, **5**, pp 347–359.

[2000] Gers F.A., Schmidhuber J., Cummins F., Continual prediction with LSTM. *Neural Computation*,**12**, (10), pp 2451–2471.

[2002] Gers F.A., Schraudolph N., Schmidhuber J., Learning precise timing with LSTM recurrent networks. *Journal of Machine Learning Research*, **3**, pp 115–143.

[2006] Giggs M.S., Maier H.R., Dandy G.C., Nixon J.B., Minimum number of generations required for convergence of genetic algorithms. in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, Vancouver, Canada, pp 2580–2587.

[1990] Girosi F., Poggio T., Networks and the best approximation property. *Biological Cybernetics*, **63**, pp 169–176.

[2004] Glasserman P., Monte Carlo methods in financial engineering, Springer-Verlag, New York, Berlin, Heidelberg.

[2010] Gosavi A., Finite horizon Markov control with one-step variance penalties. in *Conference Proceedings of the Allerton Conferences*, Allerton, IL.

[2014] Gosavi A., Solving Markov decision processes via simulation. in Handbook of Simulation Optimization, Springer.

[2014] Goudarzi A., Banda P., Lakin M.R., Teuscher C., Stefanovic D., A comparative study of reservoir computing for temporal signal processing. Working Paper, University of New Mexico and Portland State University.

[2004] Granger C., Hyng N., Occasional structural breaks and long memory with an application to the SP500 absolute stock returns. *Journal of Empirical Finance*, **11**, pp 399–421.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1976] Graves J.E., Granville's new strategy of daily stock market timing for maximum profit. Prentice-Hall, Inc.

[2007] Grau A.J., Applications of least-square regressions to pricing and hedging of financial derivatives. Ph.D. thesis, Technische Universitat Munchen.

[2005] Graves A., Schmidhber J., Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, **18**, (5-6), pp 602–610.

[2005] Graves A., Supervised sequence labelling with recurrent neural networks. PhD thesis, Technische Universitat Munchen.

[2011] Graves A., Mohamed A-R., Hinton G., Speech recognition with deep recurrent neural networks. *International Conference on Acoustics, Speech and Signal Processing*, pp 1033–1040.

[2013] Graves A., Generating sequences with recurrent neural networks. arXiv:1308.0850.

[2004] Grech D., Mazur Z., Can one make any crash prediction in finance using the local Hurst exponent idea? *Physica A: Statistical Mechanics and its Applications*, **336**, (1), pp 133–145.

[2005] Grech D., Mazur Z., Statistical properties of old and new techniques in detrended analysis of time series. *Acta Physica Polonica B*, **36**, (8), pp 2403–2413.

[2015] Greff K., Srivastava R.K., Koutnik J., Steunebrink B.R., Schmidhuber J., LSTM: A search space odyssey. Transactions On Neural Networks And Learning Systems.

[1992] Grimmett G.R., Stirzaker D.R., Probability and random processes. Oxford Science Publications, Second Edition.

[2017] GNU Scientific Library. https://www.gnu.org/software/gsl/.

[2010] Gu G-F., Zhou W-X., Detrending moving average algorithm for multifractals. *Physical Review E*, **82**, 011136, pp 1–12.

[2017] Gu S., Lillicrap T., Ghahramani Z., Turner R.E., Levine S., Q-prop: Sample-efficient policy gradient with an off-policy critic. Conference Paper at the International Conference on Learning Representations (ICLR) 2017.

[2002] Hagan P.S., Kumar D., Lesniewski A.S., Woodward D.E., Managing smile risk. *Wilmott Magazine*, pp 84–108.

[2015] Hallac D., Leskovec J., Boyd S., Network lasso: clustering and optimization in large graphs. Stanford University, arXiv:1507.00280.

[2017] Halperin I., QLBS: Q-Learner in the Black-Scholes (-Merton) worlds. NYU Tandon School of Engineerin

[2018] Halperin I., The QLBS Q-learner goes NuQLear: Fitted Q-Iteration, inverse RL, and option portfolios. NYU Tandon School of Engineerin

[2000] Hammer B., On the approximation capability of recurrent neural networks. *Neurocomputing*, **31**, (1-4), pp 107–123.

[2006] Han J., Kamber M., Data mining: Concepts and techniques. Elsevier ; Morgan Kaufmann, 2nd ed. Amsterdam, Boston, San Francisco, CA.

[2013] Han Y., Yang K., Zhou G., A new anomaly: The cross-sectional profitability of technical analysis. *Journal of Financial and Quantitative Analysis*, **48**, (5), pp 1433–1461.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2001] Hand D., Mannila H., Smyth P., Principles of data mining. MIT Press.

[1990] Hansen L.K., Salamon P., Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**, (10), pp 993–1001.

[2016] Harari Y.N., Homo deus: A brief history of tomorrow. Harvill Secker.

[2013] Hargreaves C., Hao Y., Prediction of stock performance using analytical techniques. *Journal of Emerging Technologies in Web Intelligence*, **5**, (2), pp 136–142.

[1997] Harrald P.G., Kamstra M., Evolving artificial neural networks to combine financial forecasts. *IEEE Transactions on Evolutionary Computation*, **1**, (1), pp 40–52.

[2005] Harrington N., The link between bollinger bands and the commodity channel index Harrington. *Technical Analysis of Stocks and Commodities*, **23**, (10), pp 32–38.

[1979] Harrison J.M., Kreps D., Martingale and arbitrage in multiperiods securities markets. *Journal of Economic Theory*, **20**, pp 381–408.

[1981] Harrison J.M., Pliska S.R., Martingales and stochastic integrals in the theory of continuous trading. *Stochastic Processes Applications*, **11**, pp 215–260.

[2002] Harvey C.R., Travers K.E., Costa M.J., Forecasting emerging market returns using neural networks. *Emerging Markets Quarterly*, , pp 1–12.

[2009] Hastie T., Tibshirani R., Friedman J., The elements of statistical learning: Data mining, inference, and prediction. Springer Series in Statistics, Second Edition.

[2000] Haykin S., Adaptive filter theory. Prentice-Hall, Upper Saddle River, New Jersey.

[2008] Hazan E., Kale S., Extracting certainty from uncertainty: Regret bounded by variation in costs. *COLT*, pp –.

[1949] Hebb D.O., The organization of behavior. Wiley, New York.

[1998] Helstrom T., Holmstrom K., Predicting the stock market. Published as Opuscula ISRN HEV-BIB-OP-26-SE.

[1952] Hestenes M.R., Stiefel E., Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, **49**, pp 409–436.

[1993] Heston S., A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Review of Financial Studies*, **6**, pp 327–343.

[2000] Hill J.R., Pruitt G., Hill L., The ultimate trading guide. John Wiley & Sons, Wiley Trading Advantage.

[1994] Ho T.K., Hull J.J., Srihari S.N., Decision combination in multiple classifier systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**, (1), pp 66–75.

[1995] Ho T.K., Random decision forests. in *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, Montreal, pp 14–16

[1998] Ho T.K., The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**, (8), pp 832–844

[2004] Ho D-S., Lee C-K., Wang C-C., Chuang M., Scaling characteristics in the Taiwan stock market. *Physica A*, **332**, pp 448–460.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1997] Hochreiter S., Schmidhuber J., Long short-term memory. *Neural Computation*, **9**, (8), pp 1735–1780.

[2001] Hochreiter S., Bengio Y., Frasconi P., Schmidhuber J., Gradient flow in recurrent nets: The difficulty of learning long-term dependencies. in A field guide to dynamical recurrent neural networks. IEEE Press.

[1962] Holland J.H., Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery*, **9**, pp 297-314.

[1975] Holland J.H., Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor.

[1982] Hopfield J.J., Neural networks and physical systems with emergent collective computational abilities. *Proceedings of National Academy of Sciences USA*, **79**, pp 2554–2558.

[1985] Hopfield J.J., Tank D.W., Neural computation of decisions in optimization problems. *Biological Cybernetics*, **52**, pp 141–152.

[1987] Hopfield J.J., Tank D.W., Computing with neural circuits: A model. *Science*, **233**, pp 625–633.

[1989] Hornik K., Stinchcombe M., White H., Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**, (5), pp 359–366.

[1991] Hornik K., Approximation capabilities of multilayer feedforward networks. *Neural Networks*, **4**, (2), pp 251–257.

[2008] Hsieh C-J., Chang K-W., Lin C-J., Keerthi S.S., Sundararajan S., A dual coordinate descent method for large-scale linear SVM. in *Proceedings of the 25th International Conference on Machine Learning (ICML) 08*, New York, NY, USA, pp 408–415.

[1988] Huang C., Litzenberger R.H., Foundations for financial economics. North-Holland.

[2005] Hubel D.H., Wiesel T.N., Brain and visual perception: The story of a 25-year collaboration. Oxford University Press US, pp 106.

[2017] Hudson R., Urquhart Y., Wang P., Why do moving average rules work? Comprehensive evidence from world markets.

[1951] Hurst H.E., Long-term storage capacity of reservoirs. *Trans. Amer. Soc. Civil Engineers*, **116**, pp 770–799.

[1994] Hutchinson J.M., Lo A.W., Poggio T., A nonparametric approach to pricing and hedging derivative securities via learning networks. *Journal of Finance*, **49**, (3), pp 851–889.

[2012] Ihlen E.A.F., Introduction to multifractal detrended fluctuation analysis in Matlab. *Frontiers in Physiology*, **3** (141), pp 1–18.

[2013] Ihlen E.A.F., Multifractal analyses of response time series: A comparative study. *Behav. Res.*, **45**, pp 928–945.

[2014] Ihlen E.A.F., Vereijken B., Detection of co-regulation of local structure and magnitude of stride time variability using a new local detrended fluctuation analysis. *Gait & Posture*, **39**, pp 466–471.

[2000] Ilkka T., Data is more than knowledge. *Journal of Management Information Systems*, **6**, (3), pp 103–117.

[2003] Ilonen J., Kamarainen J.K., Lampinen J., Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters*, **17**, pp 93–105.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2016] Inthachot M., Boonjing V., Intakosum1 S., Artificial neural network and genetic algorithm hybrid intelligence for predicting Thai ttock price index trend. *Computational Intelligence and Neuroscience*, **2016**, pp 1–8.

[2003] Irvine A.D., Principia mathematica (Stanford Encyclopedia of Philosophy). Metaphysics Research Lab, CSLI, Stanford University.

[2004] Ishii K., van der Zant T., Becanovic V., Ploger P., Identification of motion with echo state network. in *Proceedings of the OCEANS 2004 MTS/IEEE Conference*, **3**, pp 1205–1210.

[1967] Ivakhnenko A.G., Lapa V.G., Cybernetics and forecasting techniques. American Elsevier Pub. Co.

[1999] Jackwerth J.C., Option-implied risk-neutral distributions and implied binomial trees: A literature review. *Journal of Derivatives*, **7**, (2), pp 66–81.

[1991] Jacobs R.A., Jordan M.I., Nowlan S.J., Hinton G.E., Adaptive mixtures of local experts. *Neural Computation*, **3**, (1), pp 79–87.

[2001] Jaeger H., The echo state approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology.

[2002] Jaeger H., Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach. GMD-Forschungszentrum Informationstechnik, volume 5.

[2003] Jaeger H., Adaptive nonlinear system identification with Echo State Networks. International University Bremen.

[2004] Jaeger H., Seminar slides. Seminar Spring.

[2004] Jaeger H., Haas H., Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, **304**, pp 78–80.

[2007] Jaeger H., Echo state network. *Scholarpedia*, **2**, (9), pp 2330.

[2012] Jaeger H., Long short-term memory in echo state networks: Details of a simulation study. Technical report, Jacobs University Bremen.

[2001] Jain B.J., Pohlheim H., Wegener J., On termination criteria of evolutionary algorithms. GECCO 2001 - Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kauffmann, San Francisco.

[2016] Jamous R.A., El. Seidy E., Bayoum B.I., A novel efficient forecasting of stock market using particle swarm optimization with center of mass based technique. *International Journal of Advanced Computer Science and Applications*, **7**, (4), pp 342–347.

[2017] Janocha K., Czarnecki W.M., On loss functions for deep neural networks in classification. Working Paper, Jagiellonian University, Krakow, Poland.

[1991] Jansen D.W., de Vries C.G., On the frequency of large stock market returns: Putting booms and busts into perspective. *Review of Economics and Statistics*, **73**, (1), pp 18–24.

[2011] Jhankal N.K., Adhyaru D., Bacterial foraging optimization algorithm: A derivative free technique. Engineering (NUiCONE), Nirma University International Conference on 2011.

[2008] Jiang F., Berry H., Schoenauer M., Supervised and evolutionary learning of echo state networks. in *Proceedings of 10th International Conference on Parallel Problem Solving from Nature*, **5199** of LNCS, pp 215–224, Springer.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2012] Jizba P., Korbel J., Methods and techniques for multifractal spectrum estimation in financial time series. FNSPE, Czech Technical University, Prague.

[1992] Jordan M.I., Jacobs R.A., Hierarchies of adaptive experts. In J.E. Moody, S.J. Hanson, and R. Lippmann, editors, Advances in *Neural Information Processing Systems 4*, Morgan Kaufmann, San Francisco, pp 985–992.

[1995] Jordan M.I., Xu L., Convergence results for the EM approach to mixtures of experts architectures. *Neural Networks*, **8**, (9), pp 1409–1431.

[2013] Jordehi A., Jasni J., Parameter selection in particle swarm optimisation: A survey. *Journal of Experimental & Theoretical Artificial Intelligence*, **25**, (4), pp 527–542.

[2007] Junyou B., Stock price forecasting using PSO-trained neural networks. in *IEEE Congress on Evolutionary Computation*, pp 2879–2885.

[1996] Kaelbling L.P., Littman M.L., Moore A.W., Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, **4**, pp 237–285.

[2002] Kakade S., Langford J., Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pp 267–274.

[2011] Kamber H., Jaiwei P., Jian M., Data mining: Concepts and techniques. Morgan Kaufmann, (3rd ed.).

[2002] Kantelhardt J., Zschiegner S., Koscielny-Bunde E., Bunde A., Havlin S., Stanley H.E., Multifractal detrended fluctuation analysis of nonstationary time series. *Physica A*, **316**, (1-4), pp 87–114.

[2011] Kara Y., Boyacioglu M.A., Baykan O.K., Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange. *Expert Systems with Applications*, **38**, (5), pp 5311–5319.

[2004] Karaboga D., Okdem S., A simple and global optimization algorithm for engineering problems: Differential evolution algorithm. *Turkish Journal of Electrical Engineering & Computer Sciences*, **12**, (1), pp 53–60.

[2012] Karimi H., Yousefi F., Application of artificial neural network-genetic algorithm (ANN-GA) to correlation of density in nanofluids. *Fluid Phase Equilibria*, **336**, pp 79–83.

[1956] Kelly J.L., A new interpretation of information rate. *Bell System Technical Journal*, **35**, pp 917–926.

[1960] Keltner C.W., How to make money in commodities. Keltner Statistical Service.

[1988] Kennedy M.P., Chua L.O., Neural networks for non-linear programming. *IEEE Trans. Circuit and Systems*, **35**, pp 554–562.

[1995] Kennedy J., Eberhart C., Particle swarm optimization. in *Proceedings of the 1995 IEEE International Conference on Neural Networks*, Australia, pp 1942–1948.

[2000] Kim K-J., Han I., Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Systems with Applications*, **19**, (2), pp 125–132.

[2004] Kim K., Choi J-S., Yoon S-M., Multifractal measures for the yen-dollar exchange rate. *Journal of the Korean Physical Society*, **44**, (3), pp 643–646.

[2004b] Kim K-J., Won Boo L., Stock market prediction using artificial neural networks with optimal feature transformation. *Neural computing and applications*, **13**, (3), pp 255–260.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2006] Kim K-J., Artificial neural networks with evolutionary instance selection for financial forecasting. *Expert Systems with Applications*, **30**, (3), pp 519–526.

[2006] Kim M-J., Min S-H., Han I., An evolutionary approach to the combination of multiple classifiers to predict a stock price index. *Expert Systems with Applications*, **31**, (2), pp 241–247.

[1990] Kimoto T., Asakawa K., Yoda M., Takeoka M., Stock market prediction system with modular neural netorks. in *Proceedings of the 1990 International Joint Conference on Neural Networks*, **1**, Washington, DC, USA, pp 1–6.

[1998] Kittler J., Hatef M., Duin R., Matas J., On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**, (3), pp 226–239.

[1997] Kivinen J., Warmuth M., Exponential gradient versus gradient descent for linear predictors. *Journal of Information and Computation*, **132**, (1), pp 1–63.

[1943] Kleene S.C., Recursive predicates and quantifiers. *American Mathematical Society Transactions*, **54**, (1), pp 41–73.

[1956] Kleene S.C., Representation of events in nerve nets and finite automata. *Annals of Mathematics Studies*, **34**, Princeton University Press, pp 3–41.

[1973] Knuth D.E., The art of computer programming. Second Edition, Volume 1/Fundamental Algorithms (2nd ed.). Addison-Wesley Publishing Company.

[1996] Kohavi R., Wolpert D.H., Bias plus variance decomposition for zero one loss functions. In *Proceedings of the 13th International Conference on Machine Learning*, Bari, Italy, pp 275–283.

[1998] Kohavi R., Provost F., Glossary of terms. *Machine Learning*, **30**, pp 271–274.

[1990] Kohonen T., The self-organizing map. in *Proceedings of the IEEE*, **78**, (9), pp 1464–1480.

[1933] Kolmogorov A.N., Grundbegriffe der Wahrscheinlichkeitsrechnung. Berlin, Germany: Springer. [Transl. Foundations of the theory of probability by N. Morrison, 2nd edn. New York, NY: Chelsea, 1956].

[2013] Kristoufek L., Vosvrda M., Measuring capital market efficiency: Global and local correlations structure. *Physica A*, **392**, (1), pp 184–193.

[2010] Krollner B., Vanstone B., Finnie G., Financial time series forecasting with machine learning techniques: A survey. in *ESANN 2010 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, Bruges, Belgium.

[2006] Kucukemre A.U., Echo state networks for adaptive filtering. Master Thesis, University of Applied Sciences, Bohn-Rhein-Sieg, Germany.

[1951] Kuhn H.W., Tucker A.W., Nonlinear programming. in *Proceedings of 2nd Berkeley Symposium*. Berkeley, University of California Press, pp 481–492.

[2003] Kuncheva L.I., Whitaker C.J., Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, **51**, (2), pp 181–207.

[1992] Kurkova V., Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, **5**, pp 501–506.

[2012] Kusakci A.O., Can M., Constrained optimization with evolutionary algorithms: A comprehensive review. *Southeast Europe Journal of Soft Computing*, **1**, (2), pp 16–24.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1997] Lagoudakis M.G., Neural networks and optimization problems A case study: The minimum cost spare allocation problem. The Center for Advanced Computer Studies, University of Southwestern Louisiana.

[1997] Lam L., Suen S.Y., Application of majority voting to pattern recognition: An analysis of its behavior and performance. *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans*, **27**, (5), pp 553–568.

[2014] Lam H.K., Ekong U., Xiao B., Ouyang G., Liu H., Chan K.Y., Ling S.H., Variable weight neural networks and their applications on material surface and epilepsy seizure phase classifications. Working Paper, King's College, London.

[1980] Lambert D.R., Commodity channel index: Tool for trading cyclic trends. Commodities Magazine, 219 Parkade, Cedar Falls, IA 50613.

[2004] Lampinen J., Storn R., Differential evolution. in *New Optimization Techniques in Engineering*, G.C. Onwubolu and B. Babu, Eds., Springer-Verlag, Berlin, pp 123–166.

[2006] Landa Becerra R., Coello Coello C.A., Cultured differential evolution for constrained optimization. *Computer Methods in Applied Mechanics and Engineering*, **195**, July, pp 4303–4322.

[1961] Landauer R., Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, **5**, (3), pp 183–191.

[1984] Lane G., Lane's stochastics. *Technical Analysis of Stocks and Commodities*, pp 87–90.

[1990] Lang K.J., Waibel A.H., Hinton G.E., A time-delay neural network architecture for isolated word recognition. *Neural Networks*, **3**, pp 33–43.

[1987] Lapedes A., Farber R., Nonlinear signal processing using neural networks: Prediction and modeling. Technical Report, LA-UR87-2662, Los Alamos, New Mexico.

[1989] LeCun Y., Boser B., Denker J.S., Henderson D., Howard R.E., Hubbard W., Jackel L.D., Backpropagation applied to handwritten zip code recognition. *Neural Computation*, **1**, (4), pp 541–551.

[1991] Lee Y., Oh S., Kim M., The effect of initial weights on premature saturation in back propagation learning. *Int Jt Conf Neural Netw*, **1**, pp 65–70.

[2005] Lee R.W., Implied volatility: Statistics, dynamics, and probabilistic interpretation. *Recent advances in applied probability*, Springer, New York, pp 241–268.

[2001] Lemarechal C., Lagrangian relaxation. In Michael Junger and Denis Naddef *Computational combinatorial optimization: Papers from the Spring School held in Schloss Dagstuhl*, May 15-19, 2000. Lecture Notes in Computer Science. Berlin, Springer-Verlag, pp 112–156.

[2012] Le Roux N., Bengio Y., Fitzgibbon A., Improving first and second-order methods by modeling uncertainty. In Sra, Suvrit; Nowozin, Sebastian; Wright, Stephen *J. Optimization for Machine Learning*, MIT Press, pp 404.

[2017] Levine S., Gupta A., Achiam J., Deep reinforcement learning. Berkeley, CS 294-112.

[2017] Li F-F., Johnson J., Yeung S., Lectures. Stanford Computer Science, Stanford.

[2010] Liao Z., Wang J., Forecasting model of global stock index by stochastic time effective neural network. *Expert Systems with Applications*, **37**, (1), pp 834–841.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1993a] Lillo W.E., Loh M.H., Hui S., Zak S.H., On solving constrained optimisation problems with neural networks : A penalty method approach. *IEEE Transactions on Neural Networks*, **4**, (6), pp 931–940.

[1993b] Lillo W.E., Hui S., Zak S.H., Neural network for constrained optimization problems. *International Journal of Circuit Theory and Applications*, **21**, pp 385–399.

[2008] Lin H-T., Li L., Support vector machinery for infinite ensemble learning. *Journal of Machine Learning Research*, **9**, pp 285–312.

[2009] Lin X., Yang Z., Song Y., Short-term stock price prediction based on echo state networks. *Expert Systems with Applications*, **36**, pp 7313–7317.

[2002] Lipton A., The volatility smile problem. *Risk*, **2**, (2).

[1992] Littlestone N., Warmuth M.K., The weighted majority algorithm. University of California, Santa Cruz.

[1994] Littlestone N., Warmuth M., The weighted majority algorithm. *Info. and Computation*, **108**, (2), pp 212–261.

[2002] Liu J., Lampinen J., On setting the control parameter of the differential evolution method. in *Proc. 8th Int. Conf. Soft Computing*, pp 11–18.

[2002] Liu J., Lampinen J., On setting the control parameter of the differential evolution method. in *Proc. 8th Int. Conf. Soft Computing*, pp 11–18, Brno University of Technology, Czech Republic.

[2005] Liu J., Lampinen J., A fuzzy adaptive differential evolution algorithm. *Soft Computing*, **9**, (6), pp 448–462.

[2009] Liu B., Fernandez F.V., De Jonghe D., Gielena G., Less expensive and high quality stopping criteria for MC-based analog IC yield optimization. Working Paper, ESAT-MICAS, Katholieke Universiteit Leuven and IMSE, CSIC and University of Sevilla.

[1991] Lo A.W., Long-term memory in stock market prices. *Econometrica*, **59**, (5), pp 1279–1313.

[2008] Lo A.W., Hedge funds, systemic risk, and the financial crisis of 2007-2008. Written Testimony of A.W. Lo, Prepared for the US House of Representative.

[2008] Lo A.W., Patel P.N., 130/30: The new long-only. *The Journal of Portfolio Management*, pp 12–38.

[2010] Lo A.W., Hasanhodzic J., The evolution of technical analysis: Financial prediction from Babylonian tablets to Bloomberg terminals. John Wiley and Sons, USA.

[1995a] Longerstaey J., Zangari P., Five questions about RiskMetrics. Morgan Guaranty Trust Company, Market Risk Research, JPMorgan.

[1995b] Longerstaey J., More L., Introduction to RiskMetrics. 4th edition, Morgan Guaranty Trust Company, New York.

[1996] Longerstaey J., Spencer M., RiskMetrics: Technical document. Fourth Edition, Morgan Guaranty Trust Company, New York.

[1996] Longin F., The asymptotic distribution of extreme stock market returns. *Journal of Business*, **69**, pp 383–408.

[2001] Longstaff F.A., Schwartz E.S., Valuing American options by simulation: A simple least-square approach. *The Review of Financial Studies*, **14**, (1), pp 113–147.

[1994] Loretan M., Phillips P., Testing the covariance stationarity of heavy-tailed time series. *Journal of Empirical Finance*, **1**, pp 211–248.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2009] Lu C-J., Lee T-S., Chiu C-C., Financial time series forecasting using independent component analysis and support vector regression. *Decision Support Systems*, **47**, (2), pp 115–125.

[2006] Lu Dang Khoa N., Sakakibara K., Nishikawa I., Stock price forecasting using back propagation neural networks with time and profit based adjusted weight factors. *SICE-ICASE International Joint Conference*, Oct., Bexco, Busa, Korea, pp 5484–5488.

[1990] Luede E., Optmization of circuits with a large number of parameters. *Archiv f. Elektr. u. Uebertr.*, Band (44), Heft (2), pp 131-138.

[1984] Luenberger D.G., Linear and nonlinear programming. Addison-Wesley.

[1988] Lukac L., Brorsen B., Irwin S., A test of futures market disequilibrium using twelve different technical trading systems. *Applied Economics*, **20**, (5), pp 623–639.

[2006] Lukosevicius M., Popovici D., Jaeger H., Siewert U., Time warping invariant echo state networks Technical Report No. 2, Jacobs University Bremen.

[2007] Lukosevicius M., Echo state networks with trained feedbacks. Technical Report No. 4, Jacobs University Bremen.

[2009] Lukosevicius M., Jaeger H., Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, **3**, (3), pp 127–149.

[2010] Lukosevicius M., On self-organizing reservoirs and their hierarchies. Jacobs University Bremen, Tech. Rep, (25).

[2012] Lukosevicius M., A practical guide to applying echo state networks. Jacobs University Bremen, Germany.

[2012b] Lukosevicius M., Reservoir computing and self-organized neural hierarchies. PhD thesis, Jacobs University Bremen, Germany.

[2012] Lukosevicius M., Jaeger H., Schrauwen B., Reservoir computing trends. *Kunstliche Intelligenz*, Springer, **26**, (4), pp 365–371.

[1996] Lux T., The stable Paretian hypothesis and the frequency of large returns: An examination of major German stocks. *Applied Economics Letters*, **6**, pp 463–475.

[2012] Lye C-T., Hooy C-W., Multifractality and efficiency: Evidence from Malaysian sectoral indices. *Int. Journal of Economics and Management*, **6**, (2), pp 278–294.

[1998] Maass W., Orponen P., On the effect of analog noise in discrete-time analog computations. *Neural Computation*, **10**, pp 1071–1095.

[2002] Maass W., Natschlager T., Markram H., Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Computation*, **14**, (11), pp 2531–2560.

[2005a] Maass W., Joshi P., Sontag E.D., Computational aspects of feedback in neural circuits. Working Paper, It was published in 2007 in *PLoS Computational Biology*.

[2005b] Maass W., Joshi P., Sontag E.D., Principles of real-time computing with feedback applied to cortical microcircuit models. Conference Paper in *Advances in neural information processing systems*, January.

[2003] MacKay D.J.C., Information theory, inference, and learning algorithms. Cambridge University Press.

[1977] Mackey M.C., Glass L., Oscillation and chaos in physiological control systems. *Science*, **197**, (4300), pp 287–289.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1993] Maheswaran S., Sims C., Empirical implications of arbitrage-free asset markets. in P.C.B. Phillips, ed., *Models, Methods and Applications of Econometrics*, Cambridge, Basil Blackwell, pp 301–316.

[2011] Maknickiene N., Vytautas Rutkauskas A., Maknickas A., Investigation of financial market prediction by recurrent neural network. *Innovative Infotechmologies for Science, Bussiness and Education*, **2**, (11), pp 3–8.

[1982] Makridakis S., Andersen A., Carbon R., Fildes R., Hibon M., Lewandowski R., Newton J., Parzen R., Winkler R., The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting*, **1**, pp 111–153.

[1989] Mallat S.G., A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **11**, (7), pp 674–693.

[1990] Mallat S.G., Hwang W.L., Singularity detection and processing with wavelets. Technical Report No. 549, Computer Science Department, New York University.

[1992] Mallat S.G., Hwang W.L., Singularity detection and processing with wavelets. *IEEE Trans. Inform. Theory*, **38**, pp 617–643.

[1992b] Mallat S.G., Zhong S., Complete signal representation with multiscale edges. *IEEE Trans.*, PAMI **14**, pp 710–732.

[1993] Malliaris M., Salchenberger L., A neural network model for estimating option prices. *Journal of Applied Intelligence*, , pp 193–206.

[1960] Mandelbrot B.B., The Pareto-Levy law and the distribution of income. *International Economic Revue*, **1**.

[1963a] Mandelbrot B.B., New methods in statistical economics. *The Journal of Political Economy*, **71**.

[1963] Mandelbrot B.B., The variation of certain speculative prices. *Journal of Business*, **36**, (4), pp 394–419.

[1971] Mandelbrot B.B., When can price be arbitraged efficiently? A limit to the validity of the random walk and martingale models. *Re. Econom. Statist.*, **53**, pp 225–236.

[1979] Mandelbrot B.B., Taqqu M., Robust R/S analysis of long-run serial correlation. in *Proceedings of the 42nd Session of the International Statistical Institute*, , pp 69–104, Manila, Bulletin of the I.S.I..

[1982] Mandelbrot B.B., The fractal geometry of nature. W.H. Freeman and Company, New York.

[1997] Mandelbrot B.B., Fisher A., Calvet L., A multifractal model of asset returns. Cowles Foundation Discussion Paper No. 1164.

[2005] Manimaran P., Panigrahi P.K., Parikh J.C., Wavelet analysis and scaling properties of time series. *Phys. Rev. E*, **72**, 046120, pp 1–5.

[2009] Manimaran P., Panigrahi P.K., Parikh J.C., Multiresolution analysis of fluctuations in non-stationary time series through descrete wavelets. *Physica A*, **388**, pp 2306–2314.

[2000] Mantegna R.N., Stanley H.E., An introduction to econophysics: Correlation and complexity in finance. Cambridge University Press, Cambridge.

[1952] Markowitz H.M., Portfolio selection. *Journal of Finance*, **7**, (1), pp 77–91.

[1952b] Markowitz H.M., The utility of wealth. *Journal of Political Economy*, **60**, pp 151–158.

[1959] Markowitz H.M., Portfolio selection, efficient diversification of investments. John Wiley and Sons, New York.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2003] Matia K., Ashkenazy Y., Stanley H.E., Multifractal properties of price fluctuations of stocks and commodities. *Europhysics Letters*, **61**, (3), pp 422–428.

[2004] Matos J.A.O., Gama S.M.A., Ruskin H., Duarte J., An econophysics approach to the Portuguese stock index-psi-20. *Physica A*, **342**, (3-4), pp 665–676.

[2008] Matos J.A.O., Gama S.M.A., Ruskin H.J., Al Sharkasi A., Crane M., Time and scale Hurst exponent analysis for financial markets. *Physica A*, **387**, pp 3910–3915.

[1943] McCulloch W., Pitts W., A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**, (4), pp 115–133.

[1996] McCulloch J.H., Financial applications of stable distributions. in G.S. Maddala, C.R. Rao, eds., *Handbook of Statistics*, **14**, Elsevier, pp 393–425.

[1997] McCulloch J.H., Measuring tail thickness to estimate the stable index α: A critique. *Journal of Business & Economic Statistics*, **15**, (1), pp 74–81.

[2008] McKay B., Evolutionary Algorithms. Encyclopedia of Ecology, Seoul National University, Seoul, Republic of Korea, pp 1464–1472.

[2010] Menkhoff L., The use of technical analysis by fund managers: International evidence. *Journal of Banking & Finance*, **34**, (11), pp 2573–2586.

[1981] Merton R.C., On market timing and investment oerformance: An equilibrium theory of value for market forecasts. *Journal of Business*, **54** (3), pp 363–406.

[2007] Metghalchi M., Glasure Y., Garza-Gomez X., Chen C., Profitable technical trading rules for the Austrian stock market. *International Business & Economics Research Journal*, **6**, (9), pp 49–58.

[2012] Metghalchi M., Marcucci J., Chang Y-H., Are moving average trading rules profitable? Evidence from the European stock markets. *Applied Economics*, **44**, (12), pp 1239–1559.

[2015] Metghalchi M., Chen C., Hayes L.A., History of share prices and market efficiency of the Madrid general stock index. *International Review of Financial Analysis*, **40**, pp 178–184

[2018] Metghalchi M., Hajilee M., Hayes L.A., Return predictability and efficient market hypothesis: Evidence from Iceland. *The Journal of Alternative Investments*, **21**, (1), pp 68–78.

[2004a] Mezura-Montes E., Coello Coello C.A., Tun-Morales E.I., Simple feassibility rules and differential evolution for constrained optimization. *Third Mexican International Conference on Artificial Intelligence, MICAI, Lecture Notes in Artificial Intelligence*, pp 707–716.

[2004] Mezura-Montes E., Coello Coello C.A., A study of mechanisms to handle constraints in evolutionary algorithms. Workshop at the Genetic and Evolutionary Computation Conference, Seattle, Washington, ISGEC.

[2006] Mezura-Montes E., Velazquez-Reyes J., Coello Coello C.A., Modified differential evolution for constrained optimization. *IEEE Congress on Evolutionary Computation*, IEEE Press, pp 332–339.

[2006b] Mezura-Montes E., Coello Coello C.A., Velazquez-Reyes J., Munoz-Davila L., Multiple offspring in differential evolution for engineering design. *Engineering Optimization*, **00**, pp 1–33.

[1986] Micchelli C.A., Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*, **2**, pp 11–22.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1995] Michalewicz Z., Gnetic algorithms, numerical optimization and constraints. L. Eshelman, ed., *Proceeding of the Sixth International Conference on Genetic Algorithms*, San Mateo, pp 151–158.

[1996] Michalewicz Z., Genetic algorithms + data structures = evolution programs. Springer Verlag.

[2003] Mikosch T., Starica C., Long-range dependence effects and ARCH modeling. In *Theory and Applications of Long-range Dependence*, Birkhauser Boston, Boston, pp 439–459.

[2009] Mingguang L., Gaoyang L., Artificial neural network co-optimisation algorithm based on differential evolution. In *Second International Symposium on Computational Intelligence and Design*, pp 256–559.

[1997] Mitchell T.M., Machine learning. McGraw-Hill.

[2003] Mittelmann H.D., An independent benchmarking of SDP and SOCP solvers. *Mathematical Programming*, **95**, (2), pp 407–430.

[1999] Mladenov V.M., Maratos N.G., Neural networks for solving constrained optimization problems. Working Paper, Technical University of Sofia.

[2016] Mnih V., Badia A.P., Mirza M., Graves A., Lillicrap T.P., Harley T., Silver D., Kavukcuoglu K., Asynchronous methods for deep reinforcement earning. arXiv preprint arXiv:1605.03835.

[2012] Mohri N., Rostamizadeh A., Talwalkar A., Foundations of machine learning. MIT Press, Massachusetts, USA.

[1989] Montana D.J., Davis L., Training feedforward neural networks using genetic algorithms. *Int Jt Conf Artif Intell*, **89**, pp 762–767.

[2006] Moyano L.G., de Souza J., Duarte Queiros S.M., Multi-fractal structure of traded volume in financial markets. *Physica A*, **371**, pp 118–121.

[1990] Muller U.A., Dacorogna M.M., Olsen R.B., Pictet O.V., Schwarz M., Morgenegg C., Statistical study of foreign exchange rates, empirical evidence of a price change scaling law, and intraday analysis. *Journal of Banking and Finance*, **14**, pp 1189–1208.

[1997] Muller U.A., Dacorogna M.M., Dave R.D., Olsen R.B., Pictet O.V., von Weizsacker J.E., Volatilities of different time resolutions: Analyzing the dynamics of market components. *Journal of Empirical Finance*, **4**, pp 213–239.

[2018] Murdoch W.J., Liu P.J., Yu B., Beyond word importance: Contextual decomposition to extract interactions from lstms. International Conference on Learning Representations.

[2009] Murguia J.S., Perez-Terrazas J.E., Rosu H.C., Multifractal properties of elementary cellular automata in a discret wavelet approach of MF-DFA. *EPL Journal*, **87**, pp 2803–2808.

[1999] Murphy J.J., Technical analysis of the financial markets: A comprehensive guide to trading methods and applications. Prentice Hall Press.

[2005] Murphy S.A., A generalization error for Q-learning. *Journal of Machine Learning Research*, **6**, pp 1073–1097.

[2003] Murtagh F., Stark J.L., Renaud O., On neuro-wavelet modeling. Working Paper, School of Computer Science, Queen's University Belfast.

[1991] Muzy J.F., Bacry E., Arneodo A., Wavelets and multifractal formalism for singular signals: Application to turbulence data. *Phys. Rev. Lett.*, **67**, (25), pp 3515–3518.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1993] Muzy J.F., Bacry E., Arneodo A., Multifractal formalism for fractal signals: The structure-function approach versus the wavelet-transform modulus-maxima method. *Phys. Rev. E*, **47**, (2), pp 875–884.

[2011] Nair B.B., Sai S.G., Naveen A., Lakshmi A., Venkatesh G., Mohandas V., A GA-artificial neural network hybrid system for financial time series forecasting. *Inf Techno Mob Commun.*, **147**, pp 499–506.

[2014] Nanculef R., Frandi E., Sartori C., Allendea H., A novel Frank-Wolfe algorithm. Analysis and applications to large-scale SVM training *Inf. Sci.*, **285**, pp 66–99 .

[1990] Nelson D.B., ARCH models as diffusion approximations. *Journal of Econometrics*, **45**, pp 7–38.

[1991] Nelson D.B., Conditional heteroskedasticity in asset returns: A new approach. *Econometrica*, **59**, pp 347–370.

[2011] Neumann K., Steil J.J., Batch intrinsic plasticity for extreme learning machines. In *International Conference on Artificial Neural Networks*, pp 339–346, Springer.

[2013] Niere H.M., A multifractality measure of stock market efficiency in Asean region. *European Journal of Business and Management*, **5**, (22), pp 13–19.

[2014] Niu H., Wang J., Financial time series prediction by a random data-time effective rbf neural network. *Soft Computing*, **18**, (3), pp 497–508.

[1999] Nocedal J., Wright S.J., Numerical optimization. Sringer Verlag.

[2011] Noman N., Bollegala D., Iba H., An adaptive differential evolution algorithm. *IEEE*, pp 2229–2236.

[2005] Norouzzadeh P., Jafari G.R., Application of multifractal measures to Tehran price index. *Physica A*, **356**, pp 609–627.

[2006] Norouzzadeh P., Rahmani B., A multifractal detrended fluctuation description of Iranian rial-US dollar exchange rate. *Physica A*, **367**, pp 328–336.

[2012] Oh G., Eom C., Havlin S., Jung W-S., Wang F., Stanley H.E., Kim S., A multifractal analysis of Asian foreign exchange markets. *European Physical Journal B*, pp 85–214.

[1993] Ohlsson M., Peterson C., Soderberg B., Neural networks for optimization problems with inequality constraints: The knapsack problem. *Neural Computation*, **5**, (2), pp 331–339.

[1998] Oksendal B., Stochastic differential equations. Springer Fifth Edition.

[2003] Olson D., Mossman C., Neural network forecasts of Canadian stock returns using accounting ratios. *International Journal of Forecasting*, **19**, pp 453–465.

[2004] Onwubolu G.C., Differential evolution for the flow shop scheduling problem. in *New Optimization Techniques in Engineering*, G.C. Onwubolu and B. Babu, Eds., Springer-Verlag, Berlin, pp 585–611.

[2013] Fernando F.E.B., Freitas A.A., Rice j., Johnson C.G., New sequential covering strategy for inducing classification rules with ant colony algorithms. *IEEE Transactions on Evolutionary Computation*, **17**, (1), pp 64–76.

[2005] Oyama A., Shimoyama K., Fujii K., New constraint-handling method for multi-objective multi-constraint evolutionary optimization and its application to space plane design. *Evolutionary and Deterministic Methods for Design*.

[2002] Palmer A., Montano J.J., Redes neuronales artificiales aplicadas al analisis de datos. Doctoral Dissertation. University of Palma de Mallorca.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1998] Papageorgiou G., Likas A., Stafylopatis A., A hybrid neural optimization scheme based on parallel updates. *International Journal of Computer Mathematics*, **67**, pp 223–237.

[1985] Parker D.B., Learning logic report TR-47. MIT Press

[1997] Partridge D., Krzanowski W.J., Software diversity: Practical statistics for its measurement and exploitation. *Information & Software Technology*, **39**, (10), pp 707–717.

[2000] Pasquini M., Serva M., Clustering of volatility as a multiscale phenomenon. *European Physical Journal B*, **16**, (1), pp 195–201.

[2015] Patel J., Shah S., Thakkar P., Kotecha K., Predicting stock and stock price index movement using Trend Deterministic Data Preparation and machine learning techniques. *Expert Systems with Applications*, **42**, pp 259–268.

[2008a] Pedersen M.E.H., Chipper

eld A.J., Parameter tuning versus adaptation: Proof of principle study on differential evolution. Technical Report HL0802, Hvass Laboratories.

[2008b] Pedersen M.E.H., Chipper

eld A.J., Tuning differential evolution for artificial neural networks. Hvass Laboratories Technical Report no. HL0803.

[2010] Pedersen M.E.H., Tuning and simplifying heuristical optimization. PhD thesis, Computational Engineering and Design Group, School of Engineering Sciences, University of Southampton.

[1994] Peng C.K., Buldyrev S.V., Havlin S., Simons M., Stanley H.E., Goldberger A.L., Mosaic organization of DNA nucleotides. *Physical Review E*, **49**, (2), pp 1685–1689.

[2006] Peters J., Schaal S., Policy gradient methods for robotics. In *International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp 2219–2225.

[2008] Peters J., Schaal S., Reinforcement learning of motor skills with policy gradients. *Neural Networks*, **21**, (4), pp 682–697.

[2011c] Peters O., Optimal leverage from non-ergodicity. *Quantitative Finance*, **11** (11), pp 1593–1602.

[1990] Poggio T., Girosi F., Networks for approximation and learning. Proceedings of ZEEE, **78**, pp 1481–1497.

[1997] Polak E., Optimization: Algorithms and consistent approximations.

[2015] Polydoros A.S., Nalpantidis L., Kruger V., Advantages and limitations of reservoir computing on model learning for robot control. IROS Workshop on Machine Learning in Planning and Control of Robot Motion

[1993] Pomerleau D.A., Knowledge-based training of artificial neural networks for autonomous robot driving. in J. Connell and S. Mahadevan, (eds.), *Robot Learning*, pp 19–43, Kluwe Academic Publishers.

[1936] Post E., Finite combinatory processes, formulation I. *The Journal of Symbolic Logic*, **1**, (3), pp 103–105.

[2001] Potters M., Bouchaud J.P., Sestovic D., Hedged Monte Carlo: Low variance derivative pricing with objective probabilities. *Physica A*, **289**, pp 517–525.

[1987] Powell M.J.D., Radial basis functions for multivariable interpolation: A review. in J.C. Mason and M.G. Cox, eds. *Algorithms for Approximation*, Clarendon Press, Oxford, pp 143–167.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1992] Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P., Numerical recipes. 2nd ed. Cambridge, Cambridge University Press.

[2005] Price K., Storn R., Lampinen J., Differential evolution: A practical approach to global optimization. Springer, Heidelberg.

[1994] Puterman M.L., Markov decision processes: Discrete stochastic dynamic programming. John Wiley & Sons.

[2001] Qi M., Predicting US recessions with leading indicators vianeural network models. *International Journal of Forecasting*, **17**, pp 383–401.

[2016] Qiu M., Song Y., Predicting the direction of stock market index movement using an optimized artificial neural network model. PLoS ONE **11** ,(5): e0155133. doi:10.1371/journal.

[1993] Quinlan J.R., C4.5: Programs for machine Learning. Morgan Kaufmann, San Francisco, CA.

[1998] Quinlan J.R., Induction of decision trees. *Machine Learning*, **1**, (1), pp 81–106.

[1993] Rabemananjara R., Zakoian J., Threshold ARCH models and asymmetries in volatility. *Journal of Applied Econometrics*, **8**, pp 31–49.

[2009] Rakhlin A., Lecture notes on online learning. Draft.

[1999] Ramsey J.B., The contribution of wavelets to the analysis of economic and financial data. *Phil. Trans. R. Soc.*, **357**, pp 2593–2606.

[2017] Rani T., Kumar N., *International Journal of Advanced Research in Computer and Communication Engineering*, **6**, (5), pp 616–623.

[1995] Refenes A.N., Neural networks in capital markets. Wiley.

[1994] Refenes A.N., Zapranis A.D., Francis G., Stock performance modeling using neural network: A comparative study with regression models. *Neural Network*, **5**, pp 961–970.

[1997] Refenes A.N., Bentz Y., Bunn D.W., Burgess A.N., Zapranis A.D., Financial time series modeling with discounted least squares backpropagation. *Neurocomputing*, **14**, (2), pp 123–138.

[2008] Reinhart R.F., Steil J.J., Recurrent neural associative learning of forward and inverse kinematics for movement generation of the redundant PA-10 robot. In Learning and Adaptive Behaviors for Robotic Systems (LAB-RS), pp 35–40.

[2009] Reinhart R.F., Steil J.J., Attractor-based computation with reservoirs for online learning of inverse kinematics. Research Institute for Cognition and Robotics (CoR-Lab), Bielefeld University.

[2009b] Reinhart R.F., Steil J.J., Reaching movement generation with a recurrent neural network based on learning inverse kinematics for the humanoid robot icub. In IEEE-RAS International Conference on Humanoid Robots (Humanoids), pp 323–330.

[2011] Reinhart R.F., Reservoir computing with output feedback. Technische Fakultät, Universität Bielefeld.

[2011] Reinhart R.F., Steil J.J., A constrained regularization approach for input-driven recurrent neural networks. *Differential Equations and Dynamical Systems*, **19**, pp 27–46.

[1999] Resnick S., Samorodnitsky G., Xue F., How misleading can sample ACFs of stable MAs be? (Very!). *Ann. Appl. Prob.*, **9**, pp 797–817.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2000] Resnick S., Van Den Berg E., Sample correlation behavior for the heavy tailed general bilinear process. *Comm. Statist. Stochastic Models*, **16**, pp 233–258.

[2016] Ribeiro M.T., Singh S., Guestrin C., Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp 1135–1144.

[1995] Rice J.A., Mathematical statistics and data analysis. Thomson Information, Second Edition.

[1951] Robbins H., Monro S., A stochastic approximation method. *Ann. Math. Statistics*, **22**, pp 400–407.

[1987] Robinson A.J., Fallside F., The utility driven dynamic error propagation network. Cambridge University Engineering Department

[1956] Rochester N., Holland J.H., Habit L.H., Duda W.L., Tests on a cell assembly theory of the action of the brain, using a large digital computer. *IRE Transactions on Information Theory*, **2**, (3), pp 80–93.

[1995] Rockafellar R.T., Convex analysis. Princeton University Press.

[2012] Rodan A., Tiňo P., Simple deterministically constructed cycle reservoirs with regular jumps. The University of Birmingham.

[1967] Rogers Jr H., Theory of recursive functions and effective computability. MIT Press (1987), Reprint of the 1967 edition, Cambridge MA.

[2010] Rogers L.C.G., Tehranchi M.R., Can the implied volatility surface move by parallel shifts? *Finance and Stochastics*, **14**, (2), pp 235–248.

[2010] Rokach L., Pattern classification using ensemble methods. World Scientific, Singapore.

[2010] Roper M., Arbitrage free implied volatility surfaces. Working Paper, School of Mathematics and Statistics, The University of Sydney, Australia.

[1958] Rosenblatt F., The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, **65**, (6), pp 386–408.

[1962] Rosenblatt F., Principles of perceptrons. Spartan, Washington, DC.

[1999] Ross B.J., A lamarckian evolution strategy for genetic algorithms. In Lance D. Chambers, editor, *Practical Handbook of Genetic Algorithms: Complex Coding Systems*, **3**, pp 1–16. CRC Press, Boca Raton, Florida.

[2017] Ross A.S., Hughes M.C., Doshi-Velez F., Right for the right reasons: Training differentiable models by constraining their explanations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, IJCAI-17, pp 2662–2670.

[2002] Rubinstein M., Markowitz's portfolio selection: A fifty-year retrospective. *The Journal of Finance*, **57**, (3), pp 1041–1045.

[1986] Rumelhart D., McClelland J., Parallel distributed processing. MIT Press, Cambridge, Mass.

[1986b] Rumelhart D.E., Hinton G.E., Williams R.J., Learning internal representations by error propagation. in *Parallel distributed processing: explorations in the microstructure of cognition*, **1**, MIT Press, Cambridge, Mass, pp 318–362.

[1986c] Rumelhart D.E., Hinton G.E., Williams R.J., Learning representations by back-propagating errors. *Nature*, **323**, pp 533–536.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1994] Rumelhart D., Widrow B., Lehr M., The basic ideas in neural networks *Communications of the ACM*, **37**, (3), pp 87–92.

[2009] Russell S.J., Norvig P., Artificial intelligence: A modern approach. Prentice Hall, (3rd ed.), Upper Saddle River, New Jersey.

[1959] Samuel A.L., Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, **3**, (3), pp 535–554.

[1965] Samuelson P.A., Proof that properly anticipated prices fluctuate randomly. *Industrial Management Review*, **6**, pp 41–50.

[1969] Samuelson P.A., Merton R.C., A complete model of warrant pricing that maximizes utility. *Industrial Management Review*, pp 17–46.

[2015] Sangwan K.S., Saxena S., Kant G., Optimization of machining parameters to minimize surface roughness using integrated ANN-GA approach. in *Proceedings of the 22nd CIRP Conference on Life Cycle Engineering*, **29**, Sydney, Australia, pp 305–310.

[2005] Santana-Quintero L.V., Coello Coello C.A., An algorithm based on differential evolution for multi-objective problems. *International Journal of Computational Intelligence Research*, **1**, ISSN 0973-1873, pp 151–169.

[1994] Sarle W.S., Neural networks and statistical models. in *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, April, 1994.

[1954] Savage L.J., The foundations of statistics. Wiley, New York, second revised edition: Dover, New York, 1972.

[1990] Schapire R.E., The strength of weak learnability. *Machine Learning*, **5**, (2), pp 197–227.

[1992] Schmidhuber J., A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running network. *Neural Computation*, **4**, (2), pp 243–248.

[2007] Schmidhuber J., Wierstra D., Gagliolo M., Gomez F.J., Training recurrent networks by Evolino. *Neural Computation*, **19**, (3), pp 757–779.

[1999] Schmitt F., Schertzer D., Lovejoy S., Multifractal analysis of foreign exchange data. *Applied Stochastic Models and Data Analysis*, **15**, pp 29–53.

[2015] Schulman J., Moritz P., Levine S., Jordan J., Abbeel P., Trust region policy optimisation: deep RL with natural policy gradient and adaptive step size. University of California, Berkeley, Department of Electrical Engineering and Computer Sciences.

[2016] Schulman J., Moritz P., Levine S., Jordan J., Abbeel P., High-dimensional continuous control using generalized advantage estimation: Batch-mode actor-critic with blended Monte Carlo and function approximator returns. Published as a conference paper at ICLR 2016

[2005] Schulmeister S., Components of the profitability of technical currency trading. WIFO Working Papers, No. 263, December.

[1996] Schwager W.F., Technical analysis. Wiley.

[1995] Schwefel H.P., Evolution and optimum seeking. John Wiley & Sons.

[1995] Schweizer M., Variance-optimal hedging in discrete time. *Mathematics of Operations Research*, **20**, pp 1–32.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2008] Senol D., Ozturan M., Stock price direction prediction using artificial neural network approach: The case of Turkey. *Journal Artif Intell*, **1**, (2), pp 70–77.

[2011] Shalev-Shwartz S., Singer Y., Srebro N., Cotter A., Pegasos: primal estimated sub-gradient solver for SVM. *Mathematical Programming*, **127**, (1), pp 3–30.

[1948] Shannon C.E., A mathematical theory of communication. *Bell System Technical Journal*, **27**, (3), pp 379–423.

[1949] Shannon C.E., Communication in the presence of noise. *Proc. I.R.E.*, **37**, pp 10–21.

[1949] Shannon C.E., Weaver W., The mathematical theory of communication. University of Illinois Press, Urbana, Illinois.

[2008] Sherrod P.H., Dtreg: predictive modeling software.

[1991] Shin Y., Ghosh J., The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation. *International Joint Conference on Neural Networks*.

[1998] Shin K-S., Kim K-J., Han I., Financial data mining using genetic algorithms technique: Application to KOSPI 200. Korea Advanced Institute of Science and Technology

[2000] Shin T., Han I., Optimal signal multi-resolution by genetic algorithms to support financial neural networks for exchange-rate forecasting. *Expert Syst. Appl.*, **18**, pp 257–269.

[2002] Shipp C.A., Kuncheva L.I., Relationships between combination methods and measures of diversity in combining classifiers. *Information Fusion*, **3**, (2), pp 135–148.

[1985] Shor N.Z., Minimization methods for non-differentiable functions. Springer-Verlag.

[2012] Shynkevich A., Performance of technical analysis in growth and small cap segments of the US equity market. *Journal of Banking & Finance*, **36**, pp 193–208.

[2012] Si T., Hazra S., Jana N.D., Artificial neural network training using differential evolutionary algorithm for classification. Department of Information Technology National Institute of Technology, Durgapur West Bengal, India.

[1991] Siegelmann H.T., Sontag E.D., Turing computability with neural nets. *Applied Mathematics Letters*, **4**, (6), pp 77–80.

[2007] Siewert U., Wustlich W., Echo-state networks with bandpass neurons: Towards generic time-scale-independent reservoir structures. Internal Status Report, *PLANET Intelligent Systems GmbH*.

[2014] Silver D., Lever G., Heess N., Degris T., Wierstra D., Riedmiller M., Deterministic policy gradient algorithms. In *International Conference on Machine Learning (ICML)*, 2014.

[1996] Skalak D.B., The sources of increased accuracy for two proposed boosting algorithms. In *Working Notes of the AAAI'96 Workshop on Integrating Multiple Learned Models*, Portland, OR.

[2008] Slowik A., Bialko M., Training of artificial neural networks using differential evolution algorithm. Department of Electronics and Computer Science, Koszalin University of Technology, Koszalin, Poland.

[1989] Smith M.J., Portmann C.L., Practical design and analysis of a simple neural optimization circuit. *IEEE Trans. Circuit and Systems*, **36**, pp 42–50.

[1973] Sneath P.H.A., Sokal R.R., Numerical taxonomy: The principles and practice of numerical classification. W.H. Freeman, San Francisco, CA.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2017] Sompolinsky H., Lectures on reinforcement learning. Harvard Canvas.

[2015] Sondwale P., Overview of predictive and descriptive data mining Techniques. *International Journal of Advanced Research in Computer Science and Software*, **5**, (4), pp. 262–265.

[2008] Sorokina D., Caruana R., Riedewald M., Fink D., Detecting statistical interactions with additive groves of trees. In *Proceedings of the 25th international conference on Machine learning*, pp 1000–1007

[1961] Sprenkle C., Warrant prices as indications of expectations. *Yale Econ. Essays 1*, pp 179–232. Reprinted in Cootner (1967), pp 412–474.

[2017] Stark L., Machine learning and options pricing: A comparison of Black-Scholes and a deep neural network in pricing and hedging DAX 30 index options. Department of Finance, Aalto University School of Business.

[2004] Steil J.J., Backpropagation-decorrelation: Online recurrent learning with O(N) complexity. Neuroinformatics Group, Faculty of Technology University of Bielefeld. In *Proceedings of the IEEE International Joint Conference on Artificial Neural Networks*, **2**, pp 843–848.

[2005] Steil J.J., Memory in backpropagation-decorrelation $O(N)$ efficient online recurrent learning. In *Proceedings of the 15th International Conference on Artificial Neural Networks*, **3697** of LNCS, pp 649–654.

[2012] Stepanek J., Stovicek J., Cimler R., Application of genetic algorithms in stock market simulation. Cyprus International Conference on Educational Research (CY-ICER-2012) North Cyprus, US08-10.

[1972] Stone H.S., Introduction to computer organization and data structures. McGraw-Hill, New York.

[1995] Storn R., Price K., Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. International Computer Science Institute, Berkeley, **TR-95-012**.

[1996] Storn R., System design by constraint adaptation and differential evolution. International Computer Science Institute, Berkeley, **TR-96-039**.

[1997] Storn R., Price K., Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, **11**, (4), pp 341–359.

[2000] Storn R., Digital filter design program. FIWIZ.

[2008] Storn R., Differential evolution research: Trends and open questions. in U.K. Chakraborty, editor, *Advances in Differential Evolution*, **1**, pp 1–31, Springer-Verlag, Berlin Heidelberg.

[2014] Stosic D., Stosic D., Stosic T., Stanley H.E., Multifractal analysis of managed and independent float exchange rates. Department of Physics, Boston University. Published in 2015 in *Physica A: Statistical Mechanics and its Applications*, **428**, pp 13–18.

[1998] Struzik Z.R., Removing divergence in the negative moments of the multi-fractal partition function with the wavelet transformation. Working Paper INS-R9803, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

[1999] Struzik Z.R., Local effective Holder exponent estimation on the wavelet transform maxima tree. in *Fractals: Theory and Applications in Engineering*, eds., M. Dekking, J. Levy Vehel, E. Lutton, C. Tricot, Springer Verlag, pp 93–112.

[2000] Struzik Z.R., Determining local singularity strengths and their spectra with the wavelet transform. *Fractals*, **8**, (2), pp 163–179.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2002] Struzik Z.R., Siebes A., Wavelet transform based multifractal formalism in outlier detection and localisation for financial time series. *Physica A*, **309**, pp 388–402.

[2003] Struzik Z.R., Econophysics vs cardiophysics: The dual face of multifractality. Working Paper, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

[1999] Sun X., The Lasso and its implementation for neural networks. Department of Statistics, University of Toronto.

[1999] Sun W., Yuan Y-X., Optimization theory and methods: Nonlinear programming. Springer.

[2009] Sussillo D., Abbott L.F., Generating coherent patterns of activity from chaotic neural networks. *Neuron*, **63**, (4), pp 544–557.

[2014] Sutskever I., Vinyals O., Le Q.V., Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pp 3104–3112.

[1999] Sutton R.S., McAllester D.A., Singh S.P., Mansour Y., Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, **99**, pp 1057-1063.

[2017] Sutton R.S., Barto A.G., Reinforcement learning: An introduction. Second Edition, MIT Press, Cambridge, MA.

[1986] Sweeney R., Beating the foreign exchange market. *The Journal of Finance*, **41**, (1), pp 163–182.

[2010] Szepesvari C., Algorithms for reinforcement learning. Morgan & Claypool Publishers.

[1996] Takefuji Y., Wang J., Neural computing for optimization and combinatorics. World Scientific Publishing Co.

[1995] Tan H., Neural network model for stock forecasting. Master's Thesis, Texas Tech University.

[2009] Tan C., Financial time series forecasting using improved wavelet neural network. Master's Thesis, Aarhus Universitet.

[2007] Tanaka-Yamawaki M., Tokuoka S., Adaptive use of technical indicators for the prediction of intra-day stock prices. *Physica A*, **383**, pp 125–133.

[2014] Tang F., Tiňo P., Chen H., Learning the deterministically constructed echo state networks. International Joint Conference on Neural Networks.

[1986] Tank D.W., Hopfield J.J., Simple neural optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit. *IEEE Trans. Circuits and Systems*, CAS-33, pp 533–541.

[1981] Taqqu M.S., Self-similar processes and related ultraviolet and infrared catastrophes. In Random Fields : Rigorous Results in Statistical Mechanics and Quantum Field Theory, *Colloquia Mathematica Societatis Janos Bolya*, Vol. 27, Book 2, pp 1027–1096.

[1995] Taqqu M.S., Teverovsky V., Willinger W., Estimators for long-range dependence: An empirical study. *Fractals*, **3**, (4), pp 785–798.

[1986] Taylor S.J., Modelling financial time series. New York, John Wiley & Sons.

[2007] Taylor G.W., Hinton G.E., Roweis S., Modeling human motion using binary latent variables. in *Advances in Neural Information Processing Systems*, **19**, pp 1345–1352, MIT Press, Cambridge.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2016] Telgarsky M., Benefits of depth in neural networks. *JMLR: Workshop and Conference Proceedings 49*, pp 1–23.

[2008] Terman D.H., Izhikevich E.M., State space. Scholarpedia, http://www.scholarpedia.org/article/Phase space, Ohio State University.

[1967] Thorp E.O., Kassouf S.T., Beat the market. New York: Random House.

[2004] Timmermann A., Granger C.W.J., Efficient market hypothesis and forecasting. *International Journal of Forecasting*, **20**, pp 15–27.

[2018] Tsang M., Cheng D., Liu Y., Detecting statistical interactions from neural network weights. Conference Paper at ICLR 2018, Department of Computer Science, University of Southern California.

[2002] Tsay R.S., Analysis of financial time series. John Wiley & Sons, Hoboken, New Jersey.

[1993] Tsitsiklis J., Bertsimas D., Simulated annealing. *Statistical Science*, **8**, (1), pp 10–15.

[2006] Turiel A., Perez-Vicente C.J., Grazzini J., Numerical methods for the estimation of multifractal singularity spectra on sampled data: A comparative study. *Journal of Computational Physics*, **216**, pp 362–390.

[1936-37] Turing A., On computable numbers, with an application to the Entscheidungsproblem. in *Proceedings of the London Mathematical Society*, Series 2, (42), pp 230–265.

[1939] Turing A., Systems of logic based on ordinals. in *Proceedings of the London Mathematical Society*, (45), pp 161-228.

[1950] Turing A., Computing machinery and intelligence. *Mind*, **236**, pp 433–460.

[2016] Van den Poel D., Chesterman C., Koppen M., Ballings M., Equity price direction prediction for day trading: Ensemble classification using technical analysis indicators with interaction effects. in *2016 IEEE Congress on Evolutionary Computation* (CEC).

[1997] Vandewalle N., Ausloos M., Coherent and random sequences in financial fluctuations. *Physica A*, **246**, (3), pp 454–459.

[1998] Vandewalle N., Ausloos M., Crossing of two mobile averages: A method for measuring the robustness exponent. *Phys. Rev.*, **58**, pp 177–188.

[1998b] Vandewalle N., Ausloos M., Multi-affine analysis of typical currency exchange rates. *European Physical Journal B.*, **4**, (2), pp 257–261.

[1998c] Vandewalle N., Ausloos M., Boveroux PH., Detrended fluctuation analysis of the foreign exchange market. Working Paper.

[1982] Vapnik V., Estimation of dependencies based on empirical data. Springer-Verlag, New York.

[1989] Vapnik V., Chervonenkis A., Necessary and sufficient conditions for consistency of the method of empirical risk minimization. in Yearbook of the Academy of Sciences of the USSR on Recognition, Classification, and Forecasting, 2, Nauka (Moscow), pp 217–249.

[1992] Vapnik V., Principles of risk minimization for learning theory. AT&T Bell Laboratories.

[2000] Vapnik V., The nature of statistical learning theory. Information Science and Statistics. Springer-Verlag.

[2017] Villarrubia G., De Paz J.F., Chamoso P., De la Prieta F., Artificial neural networks used in optimization problems. *Neurocomputing*, **272**, pp 10–16.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2015a] Vinyals O., Bengio S., Kudlur M., Order matters: Sequence to sequence for sets. in a conference paper at ICLR 2015.

[2015b] Vinyals O., Fortunato M., Jaitly N., Pointer networks. In *Advances in Neural Information Processing Systems*, pp 2692–2700.

[2010] Vivekanandan P., Nedunchezhian R., A fast genetic algorithm for mining classification rules in large datasets. *International Journal on Soft Computing*, **1**, (1), pp 10–20.

[1944] Von Neumann J., Morgenstern O., Theory of games and economic behavior. Princeton: Princeton University Press, second edition: Princeton UP, 1947.

[2008] Wang W., Nie S., The performance of several combining forecasts for stock index. in *2008 International Seminar on Future Information Technology and Management Engineering*, pp 450–455.

[2009] Wang Y., Liu L., Gu R., Analysis of efficiency for Shenzhen stock market based on multifractal detrended fluctuation analysis. *International Review of Financial Analysis*, **18**, pp 271–276.

[2011] Wang Y., Wei Y., Wu C., Analysis of the efficiency and multifractality of gold markets based on multifractal detrended fluctuation analysis. *Physica A: Statistical Mechanics and its Applications*, **390**, pp 817–827.

[2011b] Wang Y., Wu C., Pan Z., Multifractal detrending moving average analysis on the US Dollar exchange rates. *Physica A*, **390**, pp 3512–3523.

[2017] Wang Y., Applications of recurrent neural network on financial time series. Quant Finance Ltd, MSc Thesis, Department of Mathematics, Imperial College London.

[1989] Watkins C.J., Learning from delayed rewards. Ph.D. Thesis, Kings College, Cambridge, UK.

[1992] Watkins C.J., Dayan P., Q-Learning. *Machine Learning*, **8**, (3-4), pp 179–192.

[2008] Wendt H., Contributions of wavelet leaders and bootstrap to multifractal analysis: Images, estimation performance, dependence structure and vanishing moments. Confidence intervals and hypothesis tests. Docteur de l'Universite de Lyon, Ecole Normale Superieure de Lyon, Traitement du Signal - Physique.

[1974] Werbos P., Beyond regression: New tools for prediction and analysis in the behavioral sciences. PhD thesis, Harvard University.

[1990] Werbos P., Backpropagation through time: What it does and how to do it. in *Proceedings of the IEEE*, **78**, (10), pp 1550–1560.

[2000] White H., A reality check for data snooping. *Econometrica*, **68**, pp 1097–1127.

[1990] Whitley D., Starkweather T., Bogart C., Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing*, **14**, (3), pp 347–361.

[2005] Wiestra D., Gomez F., Schmidhuber J., Modeling systems with internal state Evolino. In GECCO, pp 1795–1802.

[1978] Wilder W., New concepts in technical trading systems. Trend Research, Greensboro, NC.

[1989] Williams R.J., Zipser D., A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, **1**, pp 270–280.

[1990] Williams R.J., Peng J., An efficient gradient-based algorithm for online training of recurrent network trajectories. *Neural Computation*, **2**, (4), pp 490–501.

Electronic copy available at: https://ssrn.com/abstract=3307566

[1992] Williams R.J., Zipser D., Gradient-based learning algorithms for recurrent networks and their computational complexity. in Back-propagation: Theory, architectures and applications. NJ:Erlbaum, edt. Y.Chauvin and D.E.Rumelhart, chapter 13, pp 433–486.

[1992] Williams R.J., Simple statistical gradient following algorithms for connectionnist reinforcement learning. *Machine Learning*, **8**, Issue 3-4, pp 229–256.

[2012] Wilson C., Concave functions of a single variable. Mathematics for Economics, New York University.

[2011] Witkowski B., Building a solver for optimisation problems. BSc Thesis, University of Science and Technology in krakow.

[2005] Witten J.H., Frank E., Data mining: Practical machine learning tools and techniques. Morgan Kaufmann, 2 edition.

[1992] Wolpert D.H., Original contribution: Stacked generalization. *Neural Networks*, **5**, (2), pp 241–259.

[1991] Wright A.H., Genetic algorithms for real parameter optimization. in Foundation of Genetic Algorithms, ed. G. Rawlins, *First Workshop on the Foundation of Gen. Alg. and Classified Systems*, Los Altos, CA, pp 205-218.

[2008] Wyffels F., Schrauwen B., Verstraeten D., Stroobandt D., Band-pass reservoir computing. In 2008 *IEEE International Joint Conference on Neural Networks*, pp 3204–3209.

[2009] Wyffels F., Schrauwen B., Design of a central pattern generator using reservoir computing for learning human motion. Electronics and Information Systems Department, Ghent University, Belgium.

[1995] Xu L., Jordan M.I., Hinton G.E., An alternative model for mixtures of experts. In G. Tesauro, D.S. Touretzky, and T.K. Leen, editors, *Advances in Neural Information Processing Systems 7*, MIT Press, pp 633–640. Cambridge, MA, 1995.

[1996] Xu L., Jordan M.I., On convergence properties of the EM algorithm for Gaussian mixtures. *Neural Computation*, **8**, (1), pp 129–151.

[2007] Xu Y., Yang L., Haykin S., Decoupled echo state networks with lateral inhibition. *Neural Networks*, **20**, (3), pp 365–376.

[1996] Yao J.T., Poh H.L., Equity forecasting: A case study on the KLSE index. *Neural Networks in Financial Engineering, Proceedings of 3rd International Conference on Neural Networks in the Capital Markets*, Oct 1995, London, A-P.N. Refenes, Y. Abu-Mostafa, J. Moody and A. Weigend, (eds.), World Scientific, pp 341–353.

[1998] Yao J.T., Tan C.L., A study on training criteria for financial time series forecasting. Working Paper.

[2000] Yao J.T., Tan C.L., Time dependent directional profit model for financial time series forecasting. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, July 2000, Como, Italy, **5**, pp 291–296.

[2000b] Yao J.T., Tan C.L., Option price forecasting using neural networks. *Omega*, **28**, pp 455–466.

[2006] Yasuda T., Nakamura K., Kawahara A., Tanaka K., Neural network with variable type connection weights for autonomous obstacle avoidance on a prototype of six-wheel type intelligent wheelchair. *International Journal of Innovative Computing, Information and Control*, **2**, (5), pp 1165–1177.

[2015] Yosinski J., Clune J., Nguyen A., Fuchs T., Lipson H., Understanding neural networks through deep visualization. arXiv preprint arXiv:1506.06579.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2009] Yuan Y., Zhuang X-T., Jin X., Measuring multifractality of stock price fluctuation using multifractal detrended fluctuation analysis. *Physica A*, **388**, (11), pp 2189–2197.

[1900] Yule G., On the association of attributes in statistics. *Philosophical Transactions of the Royal Society of London*, **194**, pp 257–319.

[1994] Zadeh L.A., Fuzzy logic, neural networks, and soft computing. *Communications of the ACM*, **37**, (3), pp 77–84.

[2002] Zaharie D., Critical values for the control parameters of differential evolution algorithms. in R. Matousek, P. Osmera, eds., *Proceedings of MENDEL 2002, 8th International Conference on Soft Computing*, Brno University of Technology, Faculty of Mechanical Engineering, pp 62–67, Institute of Automation and Computer Science.

[2006] Zeng F., Zhang Y., Stock index prediction based on the analytical center of version space. *Advances in Neural Networks*, **3973**, pp 458–463.

[2007] Zhang X., Chen Y., Yang J.Y., Stock index forecasting using pso based selective neural network ensemble. in *International Conference on Artificial Intelligence*, pp 260–264.

[2012] Zhang H., Rangaiah G.P., An efficient constraint handling method with integrated differential evolution for numerical and engineering optimization. *Computers and Chemical Engineering*, **37**, pp 74–88.

[2012] Zhou Z-H., Ensemble methods: Foundations and algorithms. Chapman & Hall, Machine Learning & Pattern Recognition Series.

[2008] Zhu X., Wang H., Xu L., Li H., Predicting stock index increments by neural networks: The role of trading volume under different horizons. *Expert Syst. Appl.*, **34**, (4), pp 3043–3054.

[2005] Zielinski K., Peters D., Laur R., Stopping criteria for single-objective optimization. In Proceedings of the Third International Conference on Computational Intelligence, Robotics and Autonomous Systems, Singapore.

[2006] Zielinski K., Laur R., Constrained single-objective optimization using differential evolution. in *2006 IEEE Congress on Evolutionary Computation*, Vancouver, Canada, pp 223–230.

[2007] Zielinski K., Laur R., Stopping criteria for a constrained single-objective particle swarm optimization algorithm. *Informatica*, **31**, pp 51–59.

[2008] Zielinski K., Laur R., Stopping criteria for differential evolution in constrained single-objective optimization. in Chakrabotry, Ed., Advances in Differential Evolution, **143**, Springer-Verlag, Berlin, pp 111–138.

[2006] Zimmermann H.G., Grothmann R., Schaefer A.M., Tietz C., Identification and forecasting of large dynamical systems by dynamical consistent neural networks. in *New Directions in Statistical Signal Processing: From systems to brain*. MIT Press, edt S.Haykin, J.Principe, T.Sejnowski, J.McWhirter, pp 203–242.

[2003] Zinkevich M., Online convex programming and generalized infinitesimal gradient ascent. in *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, pp 928–936.

[2009] Zinkevich M., Smola A., Langford J., Slow learners are fast. In Y. Bengio, D. Schuurmans, J. Lafferty, C.K.I. Williams, and A. Culotta, editors, Advances in **Neural Information Processing Systems 22**, pp 2331–2339.

[2016] Zoph B., Le Q.V., Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578, 2016.

Electronic copy available at: https://ssrn.com/abstract=3307566

[2007] Zunino L., Tabak B.M., Perez D.G., Garavaglia M., Rosso O.A., Inefficiency in Latin-American market indices. *The European Physical Journal B*, **60**, (1), pp 111–121.

[2008] Zunino L., Tabak B.M., Figlioa A., Perez D.G., Garavaglia M., Rosso O.A., A multifractal approach for stock market inefficiency. *Physica A*, **387**, pp 6558–6566.

Electronic copy available at: https://ssrn.com/abstract=3307566